COMMODORE MAGAZINE

VOL 2
ISSUE 1

★ VIC Released
★ New CBM Software

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled by Mervyn Beamish Graphics Pty. Limited off Commodore Wordcraft disks supplied.

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

With the first issue for 1982, we commence the second volume of the COMMODORE MAGAZINE and a new format. The first volume saw a concentration on CBM products, but with the introduction of the VIC computer, we will now run articles to suit all Commodore users.

For those subscribers who commenced with this year, we will run an index of the contents of last years COMMODORE MAGAZINE and remind everyone that limited copies of back issues are still available.

There is a lot in common between the various Commodore computers with the articles printed being applicable to most users. Please drop the Editor a note if there is a topic you would like covered or an answer to a problem you may have.

The VIC has been launched without much publicity, but the impact has been tremendous. VIC's are being used in many different applications and seems every day a new idea appears for this product. But the games continue to fascinate me and there is no shortage of good cartridge games for the VIC.

Looking forward to your contributions for 1982.

table of contents

next issue:

PET User Port Cookbook
Connecting to 'The Source'
More on VIC

CBM 8023P Printer Released

The latest addition to the growing line of CBM peripherals is a bi-directional, 136 column printer with both tractor and friction feed. The 8023P is dot-matrix, and prints 150 characters per second (CPS).

The new CBM 8023P printer is designed to operate through software control, prints upper and lower case alphabetic characters, all graphic characters available with a Commodore computer, as well as user defined characters.

The 8023P conforms to IEEE interface requirements and connects directly to a Commodore computer via an IEEE cable. The 8023P will replace the 8024 printer.

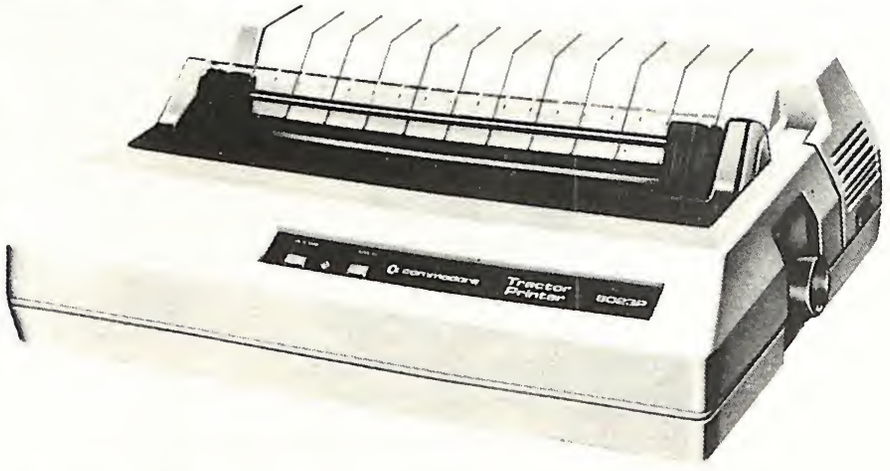
Pseudo letter quality is available by overprinting the characters and is useful where the quality of report presentation is important.

Condensed print is an option allowing column widths up to 250 characters.

Forms width is adjustable from 5 to 15 inches and up to 3 copies including the original can be made.

Because the printer is an 'intelligent' peripheral, it uses none of the computer's memory. In addition, the 8023P contains Random Access Memory (RAM), which permits storage of formatting data.

The 8023P runs with all 8000 series software packages and in most cases can be used in lieu of the 8024 printer.



'ELECTRONIC CASH BOOK' announced

Announced recently by Pittwater Computer Sales, was an Electronic Cash Book program for the Commodore microcomputer.

The Electronic Cash Book will run on the 4000 and 8000 series of Commodore microcomputers and has such features as;

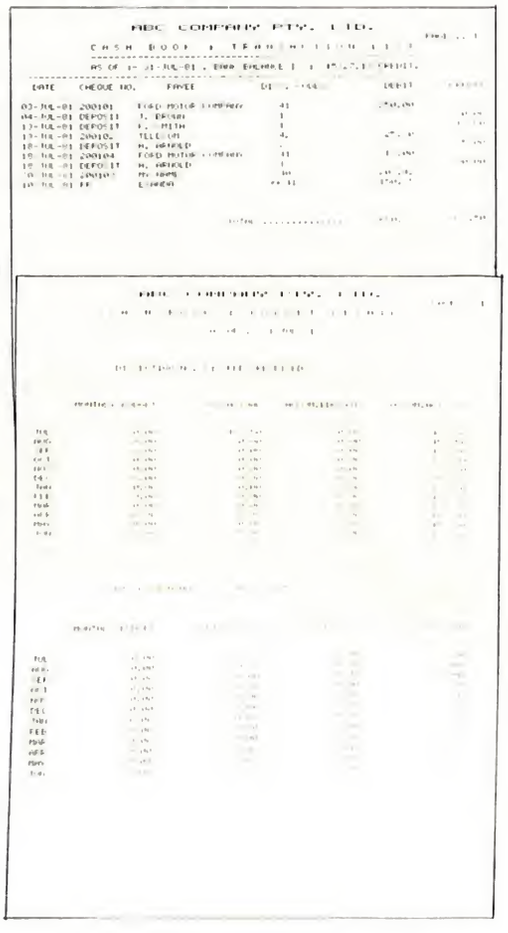
* 39 dissections of deposits
* 58 dissections of cheques
* Automatic deductions of peeriodical payments
* Completely updated Bank balance
* Reconciliation of Bank statements
* List of unreconciled cheques
* Complete enquiry facility including cheque dissections, periodical payments etc.
* Full budgetry facilities allowing real control of cash flow
* Transaction reports
* Dissection reports

Supplied with full documentation and one program diskette, any number of Companies or businesses can be configured to run on the system.

The program is complied for additional speed and is supplied with a key which will protect the information you have entered into the system.

For further information, contact your Commodore Dealer or contact

Pittwater Computer Sales
Suite 13, 9 Bungan Street
Mona Vale NSW 2103
Phone 997-4495



Sample print out from the 'ELECTRONIC CASH BOOK' — a new program recently announced by Pittwater Computer Sales.

E.B.S. Announce New Software

With one of the first business packages for the VIC 20 computer, E.B.S. recently annouced new software for the Commodore range of Equipment.

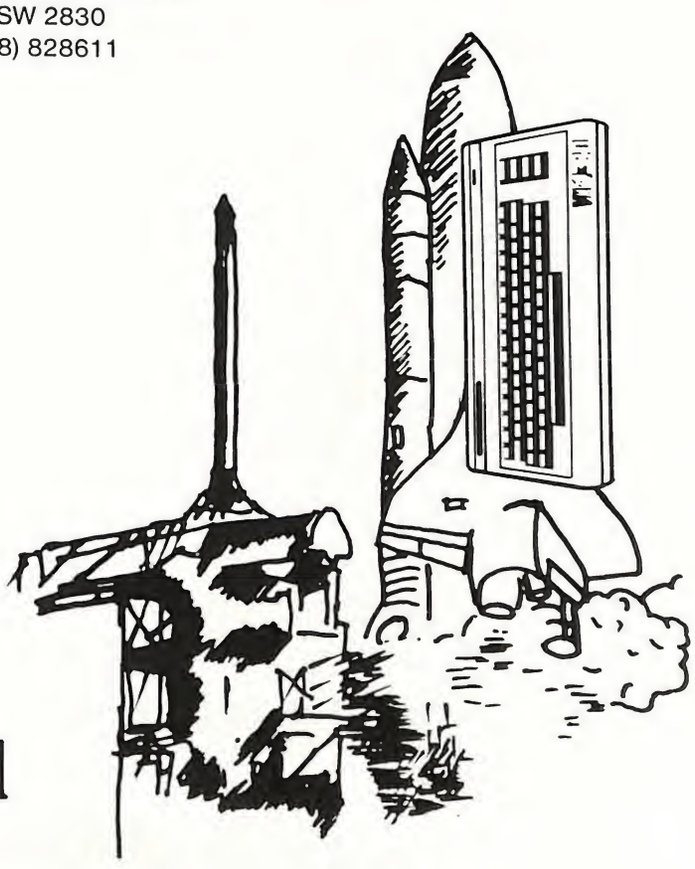
The EBS DEBTORS DATA CAPTURE SYSTEM allows information to be entered to cassette type via the VIC 20 computer and batch processed at a bureau for detailed reporting.

For Public Accountants, a low cost, efficient, Commodore system has been designed to fill the needs of the Public Accountant whose practice is small to medium sized. The EBS PUBLIC ACCOUNTANTS PACKAGE is available for immediate installation.

Parcel delivery and small transport companies have been catered for with the EBS TRANSPORT SYSTEM which is designed to meet the needs of this type of business.

Information on these packages is given on page 24 of this issue, but for further information and to arrange a demonstration, contact your Commodore Dealer or

Electronic Business Systems,
Suite 4, 118 Macquarie Street,
Dubbo NSW 2830
Phone (068) 828611



VIC 20 Launched

After much delay, the VIC 20 has been supplied to all VIC Dealers and initial reports indicate outstanding sales. Many customers have been waiting over six months, however we are informated their patience has been rewarded with an outstanding product.

At the time of delivery, Dealers were supplied with a range of VIC peripherals and accessories to compliment the VIC 20 computer. These include;

VIC Cassette – for storage and retrieval of data and programs.

VIC 1540 Disk Drive – a low cost floppy disk drive to plug into the VIC 20 computer.

VIC 1515 Printer – a low cost dot matrix printer to plug into the VIC 20 computer.

MEMORY & PROGRAMMING CARTRIDGES

Memory expansion and programming catridges are available for the VIC 20 computer.

These cartridges simply plug into the memory expansion port and are supplied with instruction booklets.

Details of these cartridges will be given in the next issue of this magazine.

VIC 1210 3K memory expansion
VIC 1110 8K memory expansion
VIC 1111 16K memory expansion
VIC 1211 VIC 20 Super Expander
VIC 1212 Programmers Aid Cartridge
VIC 1213 Machine Language Monitor

RECREATIONAL CARTRIDGES

Many games and educational programs are to be released on cartridge and include;

VIC 1901 VIC Avenger
VIC 1902 VIC Star Battle
VIC 1904 Superslot
VIC 1905 Jelly Monster
VIC 1906 VIC Super Alien
VIC 1907 Super Lander
VIC 1908 Draw Poker
VIC 1909 Road Race
VIC 1910 Rat Race

Reviews on these cartridges will commence from the next issue of the Commodore Magazine.

DISABLING THE STOP KEY

Jim Butterfield.

The STOP key can be disabled so that a program cannot be accidently (or deliberately) stopped.

METHOD A is quick. Note that any cassette tape activity will reset the STOP key to it's normal function.

BASIC 1.0 ROM:
Disable STOP with POKE 537,136
Restore STOP with POKE 537,133

BASIC 2.0 ROM:
Disable STOP with POKE 144,49
Restore STOP with POKE 144,46

BASIC 4.0 ROM:
Disable STOP with POKE 144,88
Restore STOP with POKE 144,85

VIC 20
Disable STOP with POKE 808,114
Restore STOP with POKE 808,112

Method A disconnects the computer's clock (TI and TI$). If your program needs these, use *Method B.*

METHOD B is more lengthy, but does not disturb the clocks. This method prohibits cassette tape activity.

BASIC 1.0 ROM:

```
100 r$="20>:??:9778=09024<88>6"
110 for i=1 to len(r$)/2
120 poke i+900,asc (mid$(r$,i*2-1))
*16+asc(mid$(r$,i*2))-816:next i
```

After the above has run:
Disable STOP with POKE 538,3
Restore STOP with POKE 538,230

BASIC 2.0 ROM:

```
100 r$="20>:??:9??8=9;004<31>6"
110 for i=1 to len(r$)/2
120 poke i+813,asc(mid$(r$,i*2-1))
*16+asc(mid(R$,i*2))-816:next i
```

After the above has run:
Disable STOP with POKE 145,3
Disable STOP with POKE 145,230

BASIC 4.0 ROM:

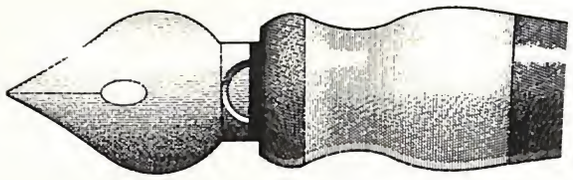
```
100 r$="20>:??:9??8=9;004<58>4"
110 for i=1 to len(r$)/2
120 poke i= 852,asc(mid$(r$,i*2-1))
*16+asc(mid$(r$,i*2))-816:next i
```

After the above has run:
Disable STOP with POKE 145,3
Restore STOP with POKE 145,228

HOW THEY WORK:

Method A skips the clock update and stop key test. *Method B* builds a small program into low memory which allows the clock update and stop key test to be performed, but then nullifies the results of this test. The small program for *Method B* is contained in R$ in "pig hexadecimal" format. Machine language programmers would read this as: 20 EA FF (do clock update, stop key test) A9 FF 8D 9B 00 (cancel stop test result) 4C 58 E4 (continue with keyboard service, etc.)

Letters to the Editor



Mr. N. Plumber,
7 Landers Road,
Lane Cove,
Sydney,
N.S.W. 2064

Dear Editor,

I was reading in one of the issues of the Commodore Magazine that it is possible to get real time delays in programs by using the WAIT command latched to the real-time clock.

I have used this with great success on our company's CBM microcomputer, because I don't need to use up variables as with FOR-NEXT loop delays. But since then I have purchaseed a VIC 20 and I'm finding tricks like this one, do not work. Can you think of a solution to my problem?

Yours faithfully,
Noal Plumber.

The problem is that Commodore Machines although they have the same structure and the same processor, they do have different Zero page memory maps. This is section of memory from $0000 to $00FF which is used by the operating system for storage of pointers, flags and counters. One of these memory locations in the VIC is used to store the real time clock (TI and TI$).

These three are 160, 161 and 162, correspond to minutes, seconds and jiffies (1/60 second). These all run up to 255 so location 161 can hold 255 seconds. So by resetting TI$ to "000000" we can use the WAIT command to give us 'real' delays.

TI$="000000":WAIT 160,1 This forces the computer to stop for 256 seconds.

TI$="000000":WAIT 161,10 This forces the computer to stop for 10 seconds.

Another use of the WAIT command is to use it to stop the program and wait for a response before continuing, rather than using the old method of continually scanning the keyboard for a response. I prefer to use the SHIFT as being the response so I scan location 654 for a 1. Some people may prefer to wait for a specific key to be pressed by scanning the key down flag at 203.

WAIT 654,1 This forces the computer to stop until the shift keys are pressed.

Dr R. Docters Van Leeuwen
C/- Mr V.C.M. Sutherland
Australian Embassy,
Koninginnegracht 23,
2514AB The Hague,
HOLLAND.
Telex : 32008
Telegraphic : AUSTEMBA

Dear Editor,

Important developments have taken place in Holland with respect to the standardization of communication between microcomputers of different brands. It is now possible to exchange programs on casette for instance between a PET, an APPLE and a TRS-80. The solution to the problem of incompatibility is called BASICODE.

If you would like to receive more detailed information on BASICODE please write to me at the above address.

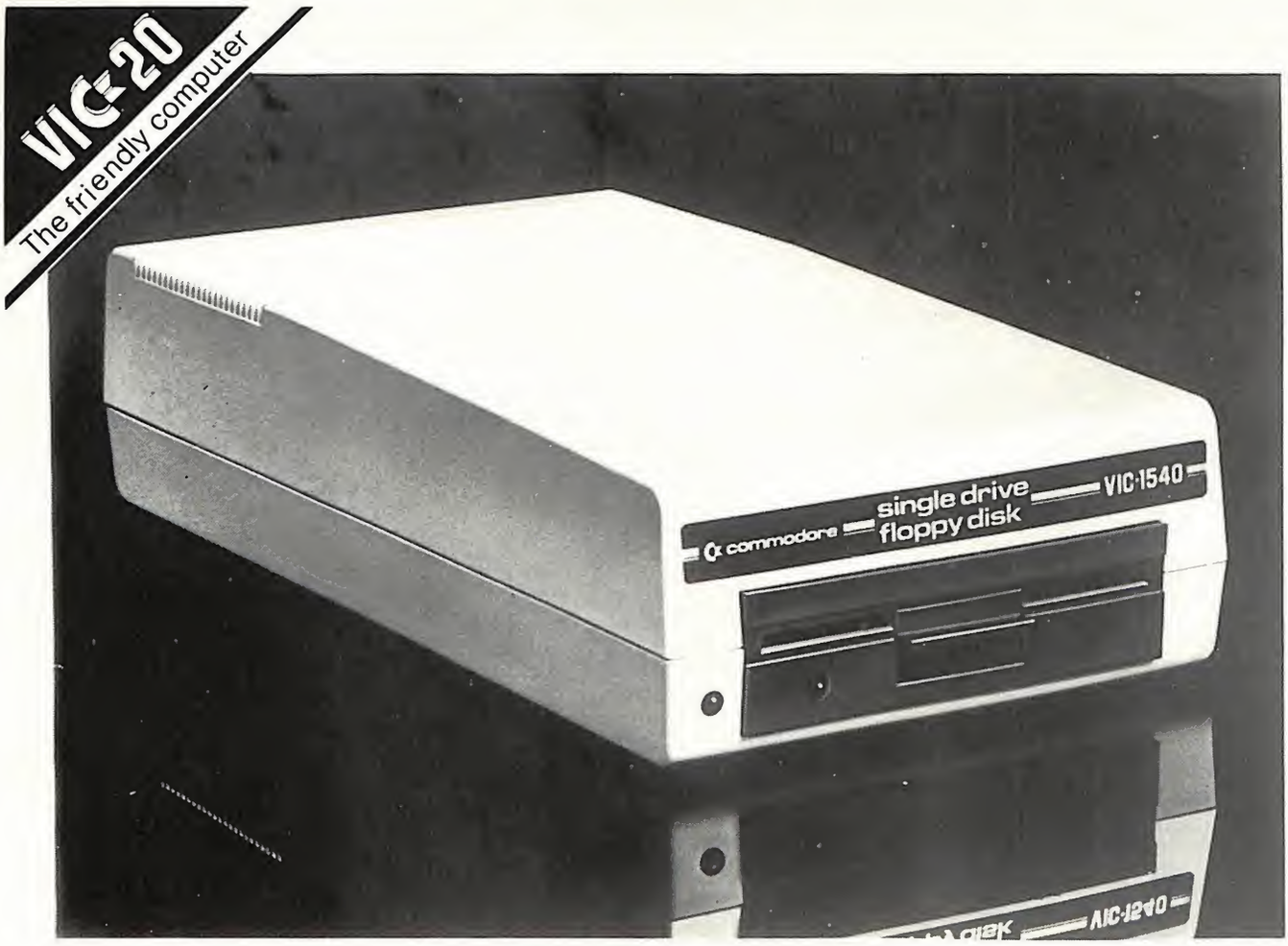
GENERAL INFORMATION ABOUT HOBBYSCOOP BASICODE

Microcomputers have the possibility to store their programs in BASIC on an audio cassette by using a regular audio cassette recorder. In order to achieve this the microcomputer contains special electronics and input and output plugs. Every brand of microcomputers has a system for coding a program in a sequence of sounds. The microcomputer can also translate these sounds into a program.

Every brand of microcomputer uses different sounds and different ways of coding. Hence it is impossible to transfer a program from one brand of microcomputer to another by means of a cassette. This means that a program which is developed on brand A and has to be used on brand B must be completely typed into a microcomputer of brand B. In a way this is medieval copying, which should not be necessary in the coming age of information.

In the Netherlands a system is developed through which programs can be exchanged between microcomputers of different brands. It is called BASICODE: a standardised sound coding system for the storage of programs in BASIC (as an ASCII – file of characters) on cassette tapes. The microcomputer is enabled to code and decode programs in BASICODE by loading it with special BASICODE translation programs developed for the:-

continued on page 32



LOW COST DATA STORAGE

Thee VIC 1540 Single Disk Drive is an inexpensive way to give the VIC fast data recovery and large capacity storage. There are at present three different DOS formats used in Commodore Disk units. The VIC single disk is fully read compatable with the existing V2A format and as such, offers all the advantages of being compatable with other Commodore computers.

The drive is a compact unit which includes an internal power supply and supporting microprocessors and memory. As with all Commodore peripherals this unit is intelligent. Commands under BASIC or with machine code handler routines can control the unit, not taking up any user memory space for buffer areas etc. This means that the disk drive can be used with a VIC 20 that only contains the minimum of 5K RAM. All processing is done in the drive unit itself, which means that other processing can take place while the disk unit is occupied. For example the drive unit is capable of VALIDATEing a disk while the controller (the VIC 20) can continue processing until the next disk access is required.

This unit is coloured an attractive off white color which will compliment the VIC 20 and other VIC perpherals and is less than 10cm in height. As the DOS (Disk Operating System) is compatable with other BASIC 2.0 computers, the unit supports program, random, sequential, and relative file structures (although the VIC 20 cannot issue the RECORD command). In general, the VIC 1540 conforms with the previous standards set on the Commmodore 4040 disk unit, storing up to 170K bytes of memory, and using DOS 2.6 which is read and write compatable with DOS 2.0 i.e. the 4040 unit.

The connection between the VIC 20 and the drive unit is via a cable connected to the serial port on the VIC 20 (not compatable with CBM microcomputers).

This unit can handle a maximum of three sequential or random files open together and using the 2K of RAM available in the DOS. The VIC 20 does not support relative files and so this type of file structure is not available to VIC users. Essentially the Drive is treated as a normal device similar to the cassette unit or the printer.

Documentation is good, a neat and concise manual is supplied with the VIC 1540 drive unit, as well as a number of utility programs supplied on a diskette and examples on how to use sequential and random file structures.

VIC 20
The friendly computer

For VIC/PET/CBM Users:

THE COMMODORE DATASETTE— SUCCESSFUL LOADING TECHNIQUES

For those of you who use a Commodore Datasette recorder, here are some helpful hints to keep those tapes rolling!

LOAD ERROR . . . a programmer's nightmare when you've just LOADed a program from your Commodore Datassette and this glaring message blinks up on your screen. What did you do wrong? Was it your program, the Datassette, the tape cassette, sunspots, poltergeists, or what?

There are several techniques which can help you load program tapes more reliably. Here are a few of them:

1. Tightening New Tapes
If you're using a new tape, either a blank tape or a pre-recorded tape (Commodore sells a variety of pre-recorded tapes), it's always a good idea to FAST FORWARD the entire tape and then REWIND it before using it. This tightens the tape on the spool and reduces the possibility of load errors due to loose tape in the cassette.

2. Positioning The Datassette
Placing the Datassette too close to your television set may expose it to interference which can produce LOAD errors. Try placing the Datassette away from the television set, and avoid coiling the Datassette cord. Commodore tests show that Datassettes placed on top of a television set produce more LOAD errors than Datassettes positioned away from the set with the cord fully extended.

3. Save: Save
When SAVEing programs on tape, it's always a good idea to save it twice. You can do this in one step. For example, if your program name is "MAGICIAN," you should SAVE it twice by typing: SAVE "MAGICIAN": SAVE "MAGICIAN" and hit RETURN. These are actually two identical commands separated by a colon. After the first MAGICIAN is SAVEd, the second command is read and the program is recorded again. Pre-recorded tapes sold by Commodore are recorded a minimum of three times so if one program is damaged or if you get a LOAD error, you can type LOAD again and get the next program on the tape.

4. Cleaning and Realigning Tape Heads
Like any tape recorder, the Datassette requires servicing to clean and demagnetize the tape heads and to adjust the head alignment. If your Datassette has been in use for some time and is producing errors, it may require servicing.

These techniques should improve your Tape Loading techniques. . . whether you're using a VIC, PET or CBM with your Commodore DATASSETTE tape recorder. ■

NEWS

OTAGO SCHOOL WINS COMMODORE CONTEST

Dunstan High School in Alexandra, Central Otago, has won the Commodore contest for secondary schools. The school will receive a Commodore PETMASTER educational system including a 4016 microcomputer with full typewriter keyboard, graphics video screen, 16K processor memory and cassette drive together with a classroom set of related manuals, textbooks and software. Total value of the prize is approximately $2000.

Dunstan High will also receive a free one-year subscription to INTERFACE, courtesy of Commodore Computer (NZ) Ltd. So will nine other schools selected by the judges. They are (not in any particular order): Glenfield College, Acukland; Waitakere College, Auckland; Tangaroa College, Auckland; Bream Bay College, Ruakaka; Riccarton High School, Christchurch; Kapiti College, Paraparaumu; Okato College, Okato; Takapuna Grammar School, Auckland; Waiau College, Tautapere, Southland

It was obvious from the entries that schools are giving a great deal of thought to the application of computers in a broad spectrum of activities, not just in the traditional maths and science areas. Several schools envisaged using computers for parent classes. Here is Dunstan High's winning entry

PROPOSAL FOR THE USE OF A COMMODORE MICROCOMPUTER SYSTEM IN A N.Z. SECONDARY SCHOOL.

The advent and spread of microprocessor technology through many aspects of our society has made it necessary for schools to prepare their pupils for a future in which computers will play an increasingly important role. This as preparation must be carried out in a manner which effectively exposes pupils to the widest possible range of computer experiences and applications. It is therefore essential to have a configuration of hardware and peripherals that enables class groups of up to 30 pupils to enjoy ready and meaningful interaction with computers. To determine what type of system is required, however, it is important to first establish which uses are relevant to the school and it's educating.

Careful planning and co-ordination could see a Commodore PETMASTER system playing a crucial role in the education of secondary pupils at all levels throughout the school year. This is possible in all of the following seven ways:-

1. Computer Awareness and Literacy

This can commence in the third form year with a short course in familiarisation (simple keyboarding, games, using a teaching program, etc.) as part of the Mathemetics course, with elementary programming skills also taught later that year to most students. In the fourth form this can be followed up with the teaching of more advanced operating skills in addition to studies on the implications of expanding computer technology for our society. Depending on the course structure, time tabling and teacher expertise at each school, these matters could be readily incorporated into Social Studies, Mathematics and/or some inter-disciplinary subject. The introduction of a full Sixth Form Certificate Computer Studies course will enable many students to continue these programs to a deeper level.

2. Programming

It is this aspect of computer usage that most readily springs to mind with many people. The teaching of programming skills, particularly to seventh form students, is an obvious and necessary use of a computer system. However, simple skills can also be taught to junior students as part of computer awareness courses – and a good proportion of these students can be expected to develop their programming expertise in their own time. No doubt also, many teachers will soon wish to learn how to program the computers they are using.

3. Word Processor

Few people can doubt that the days of typewriting are numbered, and that the keyboard skills required in the near future will have expanded into that of wordprocesssing. While some general experience of this can be given to all pupils in the junior forms, specific techniques and practice will come through the typing classes, especially in the fifth form.

4. A Teaching Machine

The possibilities for the use of computers to assist both teachers and pupils in the processes of learning are almost limitless. By means of suitable programs (either prepared commercially or by the school's own teachers) computers are able to teach and reinforce factual knowledge and concepts in every subject area and at each form level. A system of linked microcomputers can be used to teach pupils as individuals, in small groups or as a class, and can permit teacher-based control, input and direction. To give some brief examples, languages can use computers to teach vocabulary, mathematics and sciences to develop models and ideas, economics for business stimulations, English for skills practice, and so on.

5. An Assessment (or Testing) Machine

Again the computer can be used here in all subject areas and at all levels, particularly when the tests can be answered by using multi-choice, one-word or numerical formats. Also, individual, small group or class tests can be administered as the teacher requires. If it is desired, the computer can supply instant answers, scores, statistical data, and even comparisons with other tests or classes. Careful development of this aspect of computer capability will provide worthwhile data on each student's progress and success, and judicious use of this feedback can provide incentives for students towards further achievement.

6. Laboratory Apparatus

The carrying out of many demonstrations and experiments, particularly in the senior science subjects, is often impractical due to expense and danger. The use of microcomputers to stimulate such experiments is therefore very important. Radioactive decay, expansion and compression of gases, motion, and population dynamics are just a few of the many topics that can be taught and enhanced by the graphics capabilities of microcomputers. Furthermore, their use for timing, generating frequencies and constructing graphs of experimental data strengthen their place in the laboratory.

7. Administration and Data Processing

This aspect of computer usage is essentially limited to teacher-based tasks, and includes such hardy annuals as timetabling, processing of sixth form marks for accrediting and Sixth Form Certificate, sports draws, pupil records and school accounts.

Clearly the scope for using computers in the N.Z. secondary school is enormous – given the right equipment. It is also clear that the Commodore microcomputer PETMASTER system is able to provide all that is both necessary and desirable to bring each of the seven areas that have been outlined into the reach of all secondary schools. The following arrangement appears appropriate and practicable:-

a. Sufficient number of microcomputers to allow all the pupils in a class to have ready and frequent access to them.

Because a class set of 30 micros is prohibitively expensive, 10 would be economically far more realistic. This will require small groups of up to three pupils using each computer, but this is frequently an advantage as it facilitates shared input as the pupils learn. For activities which require one-to-one interaction with the computer, other related activities (bookwork, discussion, reading, etc) can usually be organised for the rest of the class.

Once the number of computers drops below 10, then the opportunity for pupils to have frequent, unpressured and worthwhile computer contact diminishes rapidly. With foresight and imaginative fundraising, it should not be beyond the scope of most secondary schools to build up a system of 10 computers within a few years. The availability of very reasonably priced microcomputers, such as Commodore PET's 4000 Series, brings this possibility even closer to practical reality.

b. The ability to link as many micros as one wishes to share storage, printing, etc.

This is absolutely essential in education for computers and, more especially, by computers in the class situation. The PETMASTER system provides this capability, enabling combinations of individual, group and class usage.

c. A printer to allow pupils to retain permanent records of their work.

d. A source of prepared and proven programs.

As the writing of programs can be a time-consuming task, and most teachers are already heavily committed, the ability to purchase quality programs which readily fit into the school computer system and subject cirricula is crucial. Commodore's comprehensive selection of such software, ranging from Microphys to Wordpro, is seen as essential to the full utilisation of the PETMASTER system.

e. The housing of the entire facility in a "dust free" room.

(i.e. no chalk) With plenty of storage space for programs, cassettes and/or diskettes and books, etc. together with ample seating and desks to provide for the wide variety of activities pupils will carry out.

f. Regular school-based in-service training by competent computer operators (both staff and pupils) to develop the skills of those with little or no experience.

Such an arrangement of Commodore computer equipment is, firstly, essential, if pupils are to experience and benefit from the many possibilities that computers offer to their education and is, secondly, practicable and affordable, given a little time and imagination.

The system itself, when fully operational, would clearly enjoy nonstop utilisation with the majority of pupils and staff regularly using it throughout the year in most subject areas and in a variety of contexts.

And that, after all, is what computer education is all about.

USING THE PRIZE COMPUTER

The computer, etc. arrived here at school safely and in excellent working order. The local paper ran an article on the arrival of the computer, generating a lot of public interest.

At present, the staff are getting to learn how to use the equipment, and are finding the "Strathclyde" program particularly useful. We have also decided to purchase another PET (16K) and a printer to help us towards the type of system outlined in our essay.

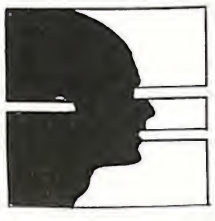
There is currently in the school a lot of enthusiasm for the use of computers in the school, and I am very grateful for your generous prize and the way in which it's arrival has enabled us to put our plans and discussions into practice.

We will also be purchasing further software and hardware next year as other funds are raised or become available.

Yours gratefully,
(On behalf of all
the staff & pupils
at Dunstan High School)

J.L. Hutton.

Photgraph shows Mr J.L. Hutton (left) accepting the prize computer from Mr S. Darnold (right).



SUPERPET BOOK SERIES

These easy-to-read, practical, tutorial/reference manuals were written to accompany an extensive software package developed to satisfy many of the educational requirements at the University of Waterloo, Ontario, Canada. Now Sams makes these valuable books available to you in a series that provides you with a complete understanding of the new Commodore SuperPET microcomputer system.

SUPERPET SYSTEM OVERVIEW©

Explains the fundamentals from hardware to Waterloo micro software. Includes information on the associated file system and Waterloo microEditor. By F. D. Boswell, T. R. Grove, K. I. McPhee, J. B. Schueler, J. W. Welch. 78 pages, 6 x 9, comb. ISBN 0-672-21903-4. © 1981. **Ask for No. 21903.** $ 7.95

SUPERPET: WATERLOO MICROFORTRAN©

Gives details of microFORTRAN, a dialect of FORTRAN designed for educational and research environments. Includes numerous examples which help familiarize you with the language, and a reference section which looks at everything from structured control statements to a FORTRAN debugger. By P. H. Dirkson and J. W. Welch. 180 pages, 6 x 9, comb. ISBN 0-672-21904-2. © 1981. **Ask for 21904.** $ 14.75

SUPERPET: WATERLOO MICROPASCAL©

An extensive look at microPascal, an interpretive implementation of the Pascal language. Discusses the versatile Waterloo microEdit program —a full-screen text editor. Simple examples help explain the language, and a reference section covers syntax and semantics. By F. D. Boswell, T. R. Grove, and J. W. Welch. 148 pages, 6 x 9, comb. ISBN 0-672-21905-0. © 1981. **Ask for No. 21905.** $ 14.75

SUPERPET: WATERLOO MICROBASIC©

A complete guide to Waterloo microBASIC, an interactive BASIC language interpreter which provides simple, comprehensive facilities for entering, running, debugging, and editing programs. By J. Wesley Graham and K. Ian McPhee. 226 pages, 6 x 9, comb.
ISBN 0-672-21906-9. © 1981.
Ask for No. 21906.

$ 14.75

SUPERPET: WATERLOO MICROAPL©

Describes this IBM/ACM standard for APL with reference to the syntax and semantics of APL statements, operators, primitive functions, I/O forms, and defined functions. By J. C. Wilson and T. A. Wilkerson. 118 pages, 6 x 9, comb. ISBN 0-672-21907-7. © 1981.
Ask for No. 21907.

$ 13.50

SUPERPET: WATERLOO 6809 ASSEMBLER©

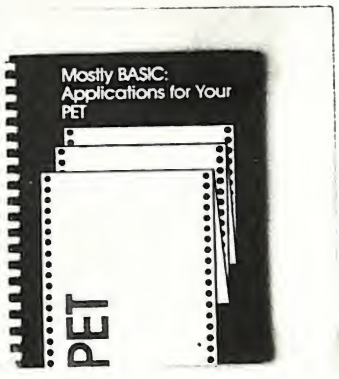
Illustrates the uses of assembly language on the Commodore SuperPET. Includes a complete reference section which covers everything from internal architecture and instruction sets to the use and functioning of the linker and monitor programs. By D. D. Cowan and M. J. Shaw. 204 pages, 6 x 9, comb. ISBN 0-672-21908-5.
© 1981. **Ask for No. 21908.**

$14.75

PET® INTERFACING

Demonstrates how you can build numerous interfacing devices for your PET® hardware. BASIC language programs are used throughout the book, so you should be familiar with this powerful programming language. The Commodore PET® microcomputer has several special-purpose interface connectors that ease the job of interfacing the computer to "real-world" hardware. Also includes a discussion of the microprocessor's internal architecture and general software/hardware interfacing. By James M. Downey and Steven M. Rogers. 264 pages, 5½ x 8½, soft.
ISBN 0-672-21795-3. © 1981.
Ask for No. 21795.

$ 22.95



MOSTLY BASIC: APPLICATIONS FOR YOUR PET®

Contains 28 useful BASIC programs for home, business, and entertainment purposes, written for use on your PET® microcomputer. Each chapter provides you with an explanation of one of the programs, a sample run, and a program listing. Programs include a telephone dialer, digital stop watch, spelling test, house buying guide, a gas mileage calculator, and others. By Howard Berenbon. 160 pages, 8½ x 11, comb.
ISBN 0-672-21790-2. © 1980. **Ask for No. 21790.**

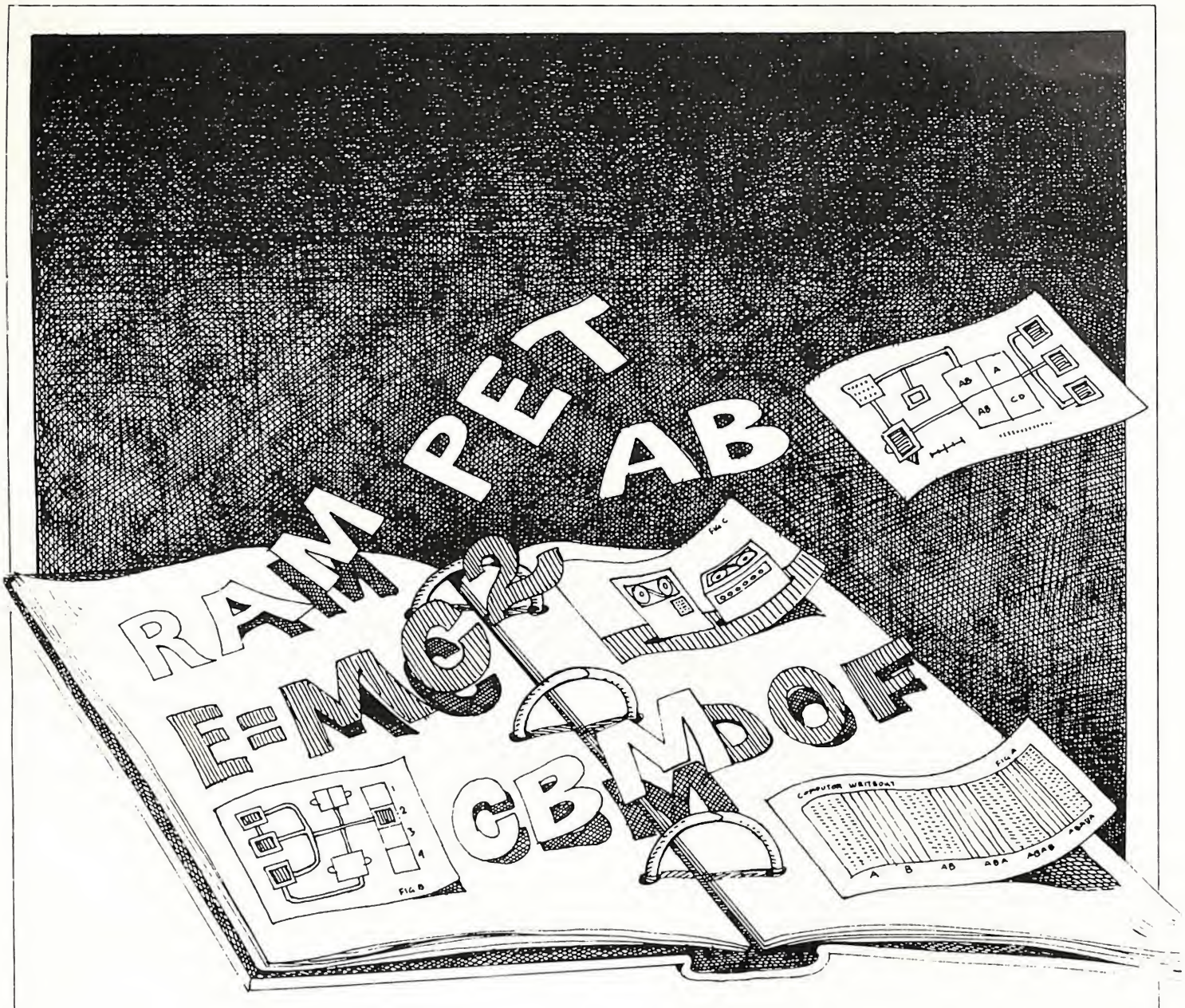
$ 17.50

AVAILABLE FROM YOUR COMMODORE DEALER

or write for catalogue to:

Prentice/Hall of Australia
7 GROSVENOR PLACE BROOKVALE N.S.W. 2100

MACHINE SPECIFIC COMPUTER BOOKS



Excerpts from a Technical Notebook

DOS 2.1 and 2.5 Bugs

The following bugs are to be found in both DOS 2.1 and 2.5:-

1. Disk Full Message Too Late

The disk operating system is supposed to check the number of blocks still free on the disk. When only two remain, the 'Disk Full' error message is supposed to be sent and the file closed, thus retaining some of the information.

What actually occurs is that the 'Disk Full' message is sent when there are no free blocks on the disk. The file cannot be closed properly. The BAM is updated to show that there are no free blocks and the file shows the number of blocks written to disk as zero. The file is also tagged as being unclosed by the use of an asterisk. A 'COLLECT' command has to be used to clear the files from disk and recover the lost blocks.

Using a save with replace causes the disk to write the new information to an unused area of the disk, close the file, and scratch the old version. When the 'Disk Full' message error is sent, output terminates and the new file does not show on the disk. The BAM is updated to show zero blocks free. The old file which was to be replaced remains untouched. A COLLECT is needed to recover the used blocks by the unclosed file.

There is no simple manner in which the lost data can be recovered and it is also difficult to test the disk before writing any information out to it without knowing the exact

size of the required file, the simplest solution is to keep a check on the amount of free space with a catalogue request and start a new disk when space starts to run short.

2. Block/Memory Commands Destroys Error Information.
Since the block and memory commands use the error channel for communicating with DOS, the last information in the error channel is always destroyed. This is not a bug. Any error checks to and from the disk should always be made straight after the disk command has been given. On the 4000/8000 series use DS for the error code and DS$ for the error message.

3. Calling The Directory From Both Drives.
If no drive is specified, the DOS returns the directory from the last drive which was accessed and then the second directory. The drive from which the directory has come is shown next to the disk title.

4. Load/Run.
Drive zero initializes and the first file is read in. If the file is not a program, an error message is generated. If the drive has already been used then the last program called up will be read into the machine. If you wish to use the quickload feature (Shift RUN/STOP), ensure the program to be run is thefirst on the directory.

5. Copying Files.
If the user sends more than four files to be copied from one drive to another, the DOS should return an error message. Under some circumstances this does not occur and the error message 'Disk ID Mismatch' will be given. Do not send more than four lines to be copied at a time. For large amounts of copying use the disk utility 'copy disk files.'

6. Copying Files on the 8050.
The copy command on the 8050 will sometimes fail after eight copies when 'COPY DO TO D1' is given, resulting in 'Illegal Track for Sector' or 'Disk ID Mismatch' as the error message. Copy also fails after eight copies when performing matched copying using 'COPY DO, ''TEST*'' TO D1,''*''.

The copy bug is not present on DOS 2.1 as the two systems are different. (DOS 2.5 has to keep bringing in the BAM for the drives while DOS 2.1 keeps both BAM's in the DOS RAM.)

7. Block Allocate on the 8050 and 4040 Disk Systems.
The Block Allocate (B-A) command is for use with the random access commands so that the disk unit knows that the block has been used and cannot be allocated to programs or sequential files. On the 3040 and 4040 drive the BAM on the disk is updated only when a file is closed. The drive should not be initialized before the BAM is written back to the disk as this actually calls the old BAM from the disk. B-A is supposed to report via the error channel whether a block has been allocated.

If the block has not been used it reports OK but if it has then the message 'NO BLOCK' is given. The error message also reports the next free track and sector. On the 3040/4040 this works correctly. On the 8050 there is a bug in DOS which can cause problems. For example, if the whole of track one has been used then DOS will report 'NO BLOCK 1 21'. This is obviously incorrect in that no sector 21 exists, i.e., DOS reports an illegal sector and will not search other tracks. Thus, if a program requires the next block it has to perform a track by track search and test for illegal sectors to determine when it has found a free block.

8. Disk Initialization on DOS 2.5.
There is a problem with the 8050 disk system at power on. The bug has the appearance of being sporadic but is very repeatable when the cause is known. For instance, the disk may not initialize, programs will not be found or the directory will not be displayed. The problem is caused by having the read head above track 55 at power on. The disk system can not recognize the problem and tries to read information from the track that it is on causing considerable confusion.

In business systems where a shift run is the method of starting the program, it is important that the head be moved to around track 39. This can be done during the system power-down at the end of the previous day, by performing an initialize via the error channel. The disk will then be in the correct state for use the next day. Note that the problem only occurs when the disk is powered down and then up again.

9. Disk Initialization DOS 2.1.
DOS 2.1 has an auto-initialize routine which relies on the disk ID changing. Thus, it is important for the user to ensure that IDs are different between disks or that the initialize command is given via the error channel. DOS 2.5 has a microswitch which will initialize the disk whenever it is opened and closed.

10. Block Write.
This command should not be used under any circumstances on DOS 2.1 because it clogs up the error channel. The only solution is to reset the disk by power down. Use the function U2 instead.

11. Disk Status DS$.
The disk error string does not monitor the commands which utilize the error channel (15). The commands associated with this are all the low-level block and memory commands such as Block-Read. The solution is to read the error from disk using:

```
10 INPUT£15,A$,B$,C$,D$,: PRINT A$,B$,C$,D$
```

12. Channel 14.
There is no channel 14 in DOS 2.x, use only those between 2-13 for normal random access. Secondary addresses 0 and 1 are used for LOAD and SAVE and should only be used under special circumstances.

Differences between old and new 4016/32s

There are potential software problems arising from the recent introduction of the 12" screen 4032 and the impending introduction of the 12" screen 4016.

Values returned by PEEK (151)
Both the 9" and the 12" screen machines use $97 (decimal 151) to hold a value related to the key being depressed. In the 9" 4016/32 this value relates to the keyboard matrix. In the 12" 4016/32 it is pseudo-ASCII (i.e. the 9" is like the 3032, while the 12" is like the 8032)

EXCEPTS FROM A TECHNICAL NOTEBOOK

4016/32 Keyboard layout

15	80	72	79	71	78	70	77	69	76	68	75	7	14	74	66	73	65
@	!	"	#	$	%	'	&		(	)							
64	33	34	35	36	37	39	38	92	40	41	95	91	93	19	17	29	20
8	64	56	63	55	62	54	61	53	60	52	59	5	12	58	50	57	49
	Q	W	E	R	T	Y	U	I	O	P		<	>	7	8	9	/
18	81	87	69	82	84	89	85	73	79	80	94	60	62	55	56	57	47
	48	40	47	39	46	38	45	37	44	36		27		42	34	41	33
	A	S	D	F	G	H	J	K	L	:				4	5	6	*
	65	83	68	70	71	72	74	75	76	58		13		52	53	54	42
	32	24	31	23	30	22	29	21	28	20				26	18	25	17
	Z	X	C	V	B	N	M	,	;	?				1	2	3	+
	90	88	67	86	66	78	77	44	59	63				49	50	51	43
					6									10	2	9	1
														0	.	-	=
					32									48	46	45	61

The figure above each character is the value returned at $97 on the 9" screen 4016/32; the figure below is that on the 12" screen 4016/32.

To check whether a program uses PEEK(151) use the FIND command in the Basic Programmers Toolkit or Basic Aid.

Top of second cassette buffer

40-column machines use 224-248 ($E0-F8) for a table of screen wrap-round flags. Because such a table became unnecessary in 80-column machines, these addresses were used for other purposes (E.G. bell, repeat, keyboard buffer length). The 12" 4016/32 has these 8032 features, but needs the table of wrap-round flags. Therefore the 12" 4016/32 uses even more of the top of the second cassette buffer than does the 8032. No programs should now use addresses between 1001 and 1024 ($03E9-0400).

Memory Map for top of second cassette buffer in 12" 4016/4032

12"	4016/32	8032		
03E9	1001	00E6	230	Repeat Key Delay Counter
03EA	1002	00E5	229	Delay between Repeat Counts
03EB	1003	00E3	227	Maximum Keyboard Buffer Size
03EC	1004	00E7	231	Bell Enable and Delay Count
03ED	1005			50 Herz Jiffy 5 Counter
03EE	1006	00E4	228	Repeat Key Flag
03EF	1007			Mask for Tabs
03F0	1008			
03F1	1009			
03F2	1010			
03F3	1011			
03F4	1012			10 bytes for Tabs
03F5	1013			
03F6	1014			
03F7	1015			
03F8	1016			
03F9	1017			
03FA	1018			User command for Monitor
03FB	1019			
03FC	1020			Timeout on IEEE
03FD	1021			
03FE	1022			
03FF	1023			
0400	1024	0400	1024	Start of Basic

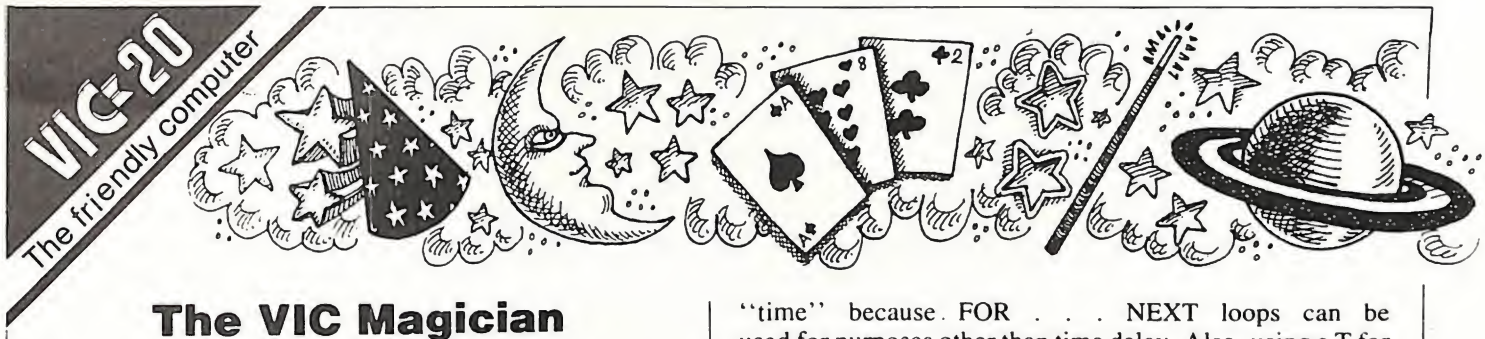
SYS 'EM!

Two useful addresses to note on BASIC 4.0 computers;

SYS 64790
SYS 54386

The first does a jump to 'warm start'... kinda like turning the machine off and back on again but without the nasty power interuption. A good way to exit your program!!

The second can be extremely handy when you want to send a M.L.M. memory dump to the printer. It seems breaking into the monitor with SYS4 cancels any CMD status you might have set up previously.



The VIC Magician

by Michael S. Tomczyk

TIME DELAY LOOPS . . . AN ADVANCED TECHNIQUE MADE EASY

In this article we want to show you an advanced BASIC programming technique which first time computerists often stumble over . . . unnecessarily. If you're just starting out in computing it's important to remember that it's just as easy to learn a so-called "advanced" technique as it is to learn a "simple" one. The problem is that good descriptions of advanced tecniques are hard to find . . . it seems like you have to read through a whole book to reach them . . . or search through a dozen computer journals for an explanation you can understand.

We forged some new ground with our innovative "PERSONAL COMPUTING ON THE VIC 20" which comes free with every computer, but there are a lot of so-called "advanced" techniques which VIC owners are ready for as soon as they finish reading their owner's guide.

One place to get advanced programming information is the VIC 20 TEACH YOURSELF PROGRAMMING SERIES, which contains a "friendly" self-teaching programming manual and some interactive tapes which lead you through the lessons, step by step.

Another good source of programming information is the VIC 20 PROGRAMMER'S REFERENCE GUIDE, which every VIC owner should own. This invaluable "bible" of the VIC covers everything from the VIC's BASIC vocabulary to machine language programming tips.

Both the TEACH YOURSELF PROGRAMMING series and the PROGRAMMER'S REFERENCE GUIDE are available through your Commodore dealer.

The Time Delay Loop . . . Special Use of "For . . . Next"
One of the best "magic" tricks which programmers use to control the speed of their programs is called the "time delay loop." This is a simple line which you put in your BASIC program to make it move at a given speed. The technique is simple. All you do is include a line which says:

FOR T = 1 TO 1000:NEXT

You can include the line anywhere in your program, wherever you want a "time delay" and you can include several delays in different places if you want. For example, the first program below PRINTs two messages, separated by a "time delay."

The T in the time delay line can be any letter, two letters, or a letter and a number, but we usually use a T to specify "time" because FOR . . . NEXT loops can be used for purposes other than time delay. Also, using a T for time makes it easy to spot the time delay loops when you list a program with a lot of FOR . . . NEXT loops used for different purposes.

Another changeable item in the time delay loop is the number 1000. This can be *any number*. A larger number makes the time delay longer and a shorter number shortens the time delay. Actually, what you're doing is telling the VIC to count to 1000 (or whatever number) before proceeding. If the number is large, the VIC takes a longer time to count than if the number is short.

EXAMPLE 1 . . . TIME DELAY

```
10PRINT"THE VIC 20 IS GREAT!"
20FOR T = 1 TO 1000: NEXT
30GOTO10
```

This prints the message, "THE VIC 20 IS GREAT!," counts to 1000, and goes back to line 10 to print the message over again. The time delay specifies how long the VIC should wait before printing the message over again. Try substituting X or A2 for the "T" in line 20 and you'll see that this doesn't change the program. Try putting another number (200 or 2000) instead of 1000 in line 20 and see how the program gets faster or slower.

Time delay loops can be used to lengthen or shorten the duration of musical or sound effect tones being played on the VIC, as shown in the following example:

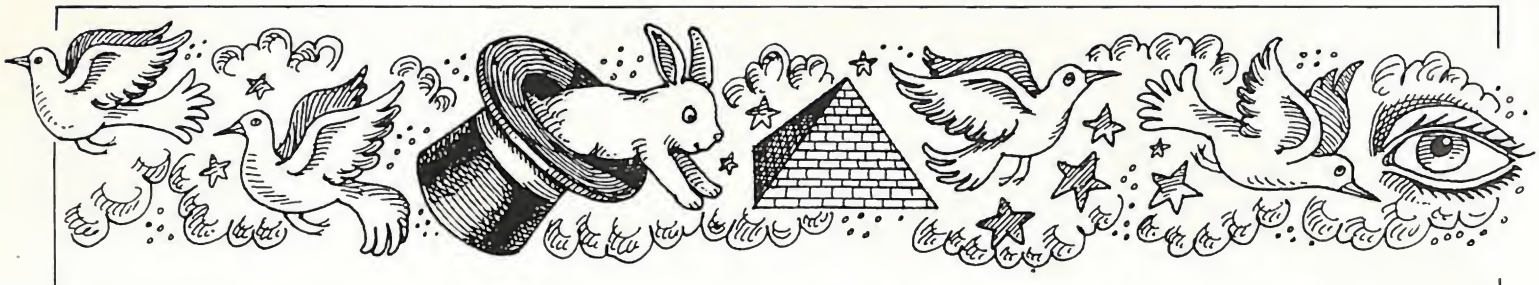
Example 2 . . . Time Delay Loops With Music
Here's a program which uses a time delay loop with a musical sound effect. The time delay relates to the length of time each note is played. You might pay special note to line 30 as well, which uses a FOR . . . NEXT . . . STEP statement to "step" through a range of VIC musical note values (from the Table of Musical Notes in the VIC user manual). Although we're looking mainly at time delay loops, there are other uses for the FOR . . . NEXT statement which we will cover in a future article. Back to time delays . . . here is the music program:

5 PRINT"WATER FILLING UP"
Prints this message on the screen while the sound effect is playing.

10 V=36878:S=36878
Set the volume equal to V and the speaker we want to use (in this case 367876) equal to S1.

20 POKEV, 15
Set the volume at maximum level (15)

30 FORN=195TO225 STEP 1:
VIC speakers can accept note values from 128 to 255. Here we are saying, for note values from 195 to 225, POKE Speaker 1 with those values, STEPping up one



at a time from note to note.

40 FORT=1TO100:NEXTT

This says count to 100 before moving to line 50 (where we play the next note). This is our TIME DELAY LOOP.

50 NEXTN

POKE the "NEXT N" into Speaker 1 . . . keep doing this until we reach the limit (which we set at 225 in line 30).

60 POKES, 0

Turn off the speaker, otherwise it will keep playing.

In this example, we see how a time delay loop affects the duration of a series of musical notes we want to play. If you want the notes to be shorter, change the "100" in line 40 to a smaller number. OR . . . use a larger number to make the notes play longer. Notice that if you make the notes VERY short (change 100 to 2 in line 40) you get interesting sound effects.

You can get "reverse" sound effects by changing line 30 to: 30 FORN=225TO195STEP−1:POKES,N This reverses the notes and steps backwards (−1) from 225 to 195 for a "water emptying" sound.

The key lesson here is the FOR . . . NEXT loop. In EXAMPLE 2 above we used the loop for two purposes . . . first, a time delay loop to extend the duration of notes we are playing and secondly, we used the FOR . . . NEXT loop to define N as a *series of note values*, then instructed the VIC to play that series one at a time by STEPping from value to value. The NEXT N in line 50 was the place where the note was actually played. You can prove that the note is actually played when the program hits line 50 by adding this line to your program:

45 PRINT"PLAY NOTE"

The program will now print "PLAY NOTE" just *before* it plays each note because you inserted the PRINT instruction before the NEXT N (next note) instruction in your program. This is important because it illustrates how you can combine sound effects with printed information (or graphic symbols) in your programs. Just mix and match PRINT statements with sound effects and you're writing programs that "sing."

Conclusion

We've taken a quick look at two major uses for time delay loops . . . one to place a "time delay" between two parts of a program to make it run slower . . . the other to place durations in a musical program where the time delay affects the sound being produced.

The best way to use time delay loops is to experiment. The best way to find out if time delays are required by your program is to see how fast it runs. If it runs too fast, slow it down by inserting a time delay. You can put a time delay loop anywhere you can put an ordinary program line, and you can use as many time delays as needed in a single program. You can even include a time delay as a GOSUB routine and keep coming back to it if you have a long-running program which requires the same delay to be repeated several times.

More information on time delays is available in the VIC owner's manual, the VIC 20 PROGRAMMER'S REFERENCE GUIDE, and most books on BASIC for the PET/CBM microcomputer.■

VIC-20 The friendly computer

GRAPPLING WITH GRAPHICS

by Garry Mason

In this article I hope to explain how the VIC 20 can be made to acheive brilliant graphics displays as used in those fantastic arcade games such as those already been written for the VIC. Although mere mortals like myself cannot produce such effects as fast in BASIC, I can certainly reveal some of the methods used to ZAP !!! those aliens for instance. To do this I need to explain how the VIC and its character generator works, but more importantly how we can use these features from within BASIC. Let's start by finding out what a Character Generator is.

As a memory mapped screen, the VIC can display up to 255 different characters (although half are the reverse field equivilent of the other half) in any one of the 506 memory locations that corresponds to the actual character space on the screen. For each of these character positions, there is a memory location that stores a number which represents which of the characters is to be displayed in any particular place on the screen. These numbers do not directly control what any character looks like, they only give an indication as to which bit patterns the VIC can use to build up that coded character. (This is getting complicated..). These bit patterns control what each character looks like – smart, hey ?

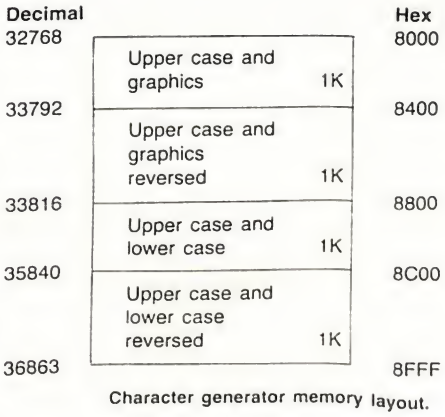
There are 127 of these bit patterns in one set of characters, one bit pattern for each possible number that can be stored in any of the 506 memory locations devoted to screen slavery. The VIC being a generally 'beaut' machine allows you to choose between two of these sets of 127 bit patterns or as the profressionals would say, character sets. These are uppercase/full graphics and lower case/upper case with limited graphics. These are located on the key board by pressing the Commodore logo key and the shift character key simultaniously.

This is where we depart the beaten track, trodden by the other Commodore Computers and many others. Here in the wonderful world of VIC we have (at no extra cost) two additional character sets. At power up these are used to form reverse field or inverse characters. This can be seen simply by moving the cursor over a character. Watch it flash between reverse video and normal, unlike some machines, the VIC is not just saying to the video circuit as on other Commodore machines:-

- Display this reverse field, then in normal again .....

Although we do not notice, the VIC is infact displaying or switching between two different characters in different character sets. First the normal character then the other in the third or fourth character set. This is the task of the extra two character sets. Only one is in use at any one time, there being one inverse set for each of the other two character sets. So there is a possibility of being able to use four character sets for our own use and still retain the other two character sets for normal characters.

Another important difference between the VIC and several other computers, including Commodore's range of business machines, is the fact that all these character bits are not locked up within the machine, but are memory mapped as well. This combined with a useful register that is used to point to the start of the character generator, makes for fun and great application programming. This system has the advantage of allowing VIC user/programmers to adjust or completely re-model the whole character set.



Character generator memory layout.

When the VIC is turned on, a register in the VIC chip is set up to point at the address in memory where the character generator is (between 32768 and 36864). At this point we cannot make any changes to the character set as these addresses are in ROM. This is 'pro' talk for Read Only Memory and as it says read only – no changes. When this register is moved either by the VIC, or by ourselves, then funny things start happening.

When we switch between upper case/graphics and lower case/upper case either by pressing the shift and Commodore logo keys simultaneously, or by PRINTing CHR$(14) or POKEing 36869, 242, this register at 36869 changes to point to the lower case/upper case character set. We know this because that is the function of the last example to change to contents of this register. This method of changing character set is not recommended as it is not easy to spot when debugging programs, especially if the displays comes to resemble a cloth cap. It is much easier to PRINT " ";CHR$(14) and be sure of the consequences, as misguarded POKEs can do nasty things to internal workings :-

try POKE 198, 20
or POKE 36869, 35

When we want to start defining our own characters will need to move the character generator into RAM. This is Random Access Memory, the opposite of Read Only Memory. Once we have moved all the bit patterns down into this section of memory and changed some pointers then the character set is at our mercy. To do this, first we need to find a protected area of memory to which we can move all the bit patterns to. Then once it is moved, the pointers in the VIC chip at 36869 can be changed to point at our copy of the character generator (in RAM).

If we are using the VIC with the 3K RAM cartridge, which is best as we need some extra room to store the duplicate character generator, it is a good plan to put the copy right at the limit of memory. This should then allow you to be confident in typing in programs of reasonable length without corrupting the character generator. Also in the interest of saving space and for simplicity we will locate our copy to 5120 dec, leaving room for one complete character set before we

run out of memory. But first it is necessary to move a few pointers around so it is impossible for a program to over-write the copy.

The two pointers we need to change are limit-of-memory and string-storage pointers. Normally these two point to the same place when no variables are being used. Type :-

```
POKE 51, 0 : POKE 52, 19 : POKE 55, 0 :
POKE 56, 19 : CLR
```

We have set up these two pointers, first string storage then limit of memory to within 256 bytes of our character generator. Notice that there are two POKEs for each of the two pointers. This is because that any address in a computer can then be broken down first into a hexadecimal number and then into a high and low hexadecimal address.

This is the chain of advents ,

$4864_{10} = 1300_{16}$

$13_{10} = 18_{16}$ and $00_{10} = 00_{16}$ (high / low)

It is quite simple to move the character generator and form our own copy as it is a simple matter of duplication. This sample program finds out what is stored in the original character generator,-

```
FOR C = 0 TO 235 ° 8
P = PEEK (C + 32768)
POKE (5120 + C), P
NEXT C
```

-then moves and places that value in the corresponding location in our copy. Now we can use this copy of the character generator but first the final touch, move the VIC pointer. This is simply acheived (thank goodness). Just poke the new starting address into location 36869 in the VIC chip.

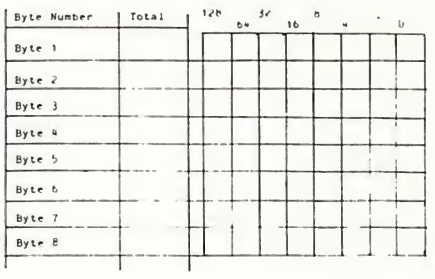
```
POKE 36869, 253 : POKE 36867,
PEEK (36867) OR 128
```

Now let's have some fun; try this little character. The POKEs replace the original character with a little face. The face can now be accessed via the appropiate key.

```
POKE 5376, 129: POKE 5377, 90:
POKE 5378, 60: POKE 5379, 102
POKE 5380, 126: POKE 5381, 60:
POKE 5382, 36: POKE 5383, 195
```

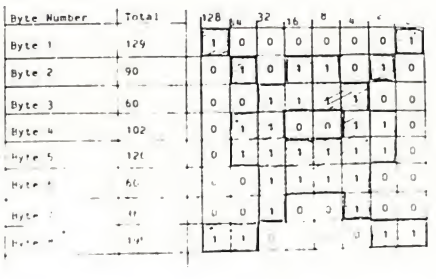
This character can, of course, be displayed in any colour just by pressing the control key and one of the colour control keys, thus changing the colour of the cursor. The only limit being that it cannot be displayed in reverse video, because of the different character that the VIC would look up by the VIC.

There are eight bytes in each character, one for each of the 8 dot lines that make up the 8 by 8 matrix of a character. To design your own character simply layout this sort of form :-



Form A).

When you are satisfied with the character, add up the column keys for each of the horizontal lines. Once this is done there should be eight coded bytes. These when 'POKE'ed into the appropriate bit pattern will transform all of those particular characters. The completed form may look something like this :-



Form B).

the VIC. Use reverse field for your definable characters and leave the other set untouched. Well, that's a good idea. All this means is that we need to bring down more of the character set than expected. This could be time consuming so it may be a good idea to use the monitor to save the finished program, thus saving the character set along with the program.

This is a good idea, these definable characters, too good to leave it all to humans. So here are a few routines to help you decode a character from within a program. In these examples, I have used A(X, Y) to represent the 8 x 8 dot mattrix. These two routines take in and read out the character from the array A(X, Y).

Now that we have learnt how to form our own characters, we then have to calculate the address of the character we wish to over-lay. This involves binary numbers and a lot of fiddling about so :-

```
A$ = "Z": REM OVERLAY CHARACTER
PRINT " "; A$
A = 5120 + (PEEK(7680)°8)
PRINT " OVERLAY STARTS AT ";A;
                    " DECIMAL "
```

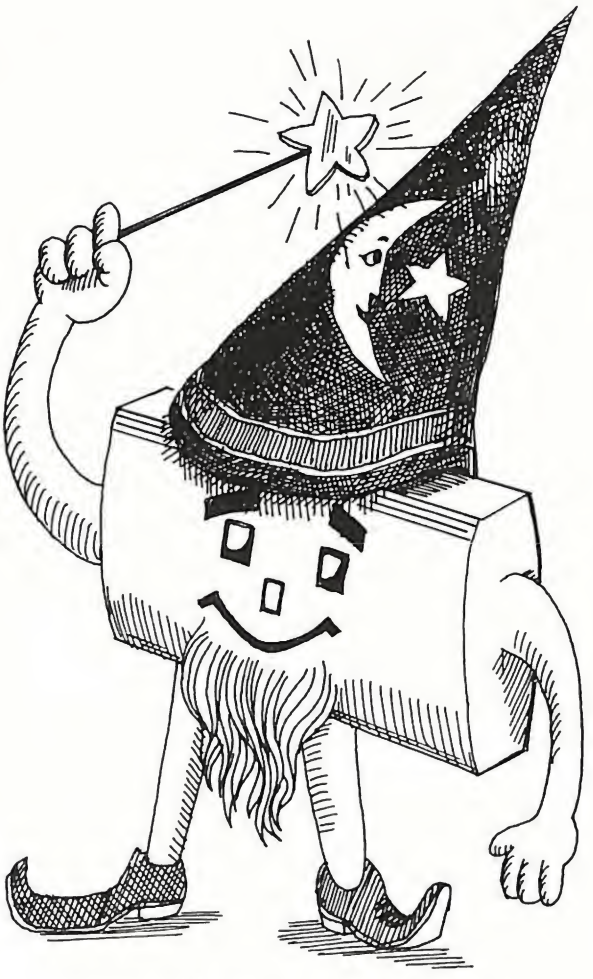
Briefly we base the address by the start location of the character generator (5120) and indexed it by the POKE code number (that stored in the memory mapped screen locations) of that particular character left shifted three times (multiply by 8). This gives us the first address for the first byte as denoted on the above form. The other seven bytes are easily found, just add +1 to each subsequent number or byte.

This opens up great possibilities – Space Invaders in BASIC running on the VIC's extra fast operating system. All sorts of things, designing special characters which can then be implimented on the VIC printer.

Using the method above would be alright if only a few characters need to be displayed, "But, what if we want to use the full 127 normal characters as well ?", I here you ask. 'Simple', says

PROGRAMMABLE CHARACTERS

```
100 REM PROGRAMMABLE
110 REM CHARACTERS.
120 REM 19TH MAR 82
130 REM ------------
140 P=PEEK(36869)
150 IF P=253THEN180
160 GOSUB330 :REM SET UP POINTERS
170 CLR:GOSUB380 :REM MOVE CHR. SET AND MAKE LIVE
180 PRINT"□1.CHANGE CHAR
190 PRINT"2.LOAD CHARSET
200 PRINT"3.SAVE CHARSET
210 PRINT"4.END
220 GETA$:IFA$<"1"ORA$>"4"THEN220
230 ONVAL(A$)GOTO250,1630,1520,240
240 END
250 GOSUB530 :REM SET UP AND START
255 REM ----------------
260 FORI=0 TO 7
270 A=B(I)
280 P=5120+C*8+I
290 POKE P,A
300 NEXT I
310 RUN180
320 REM FIX POINTERS
325 REM ----------------
330 POKE 51,0
340 POKE 52,19
350 POKE 55,0
360 POKE 56,19
370 RETURN
380 REM MOVE ROM CHR. SET INTO RAM
382 REM ------------------
385 H=256:S=32768
390 M=19*H+32*8
400 FOR I=0 TO H*8-1
410 P=PEEK(I+S)
420 POKE M+I,P
430 NEXT I
440 REM FIX VIC CHIP REGISTER
445 REM ------------------
450 P=PEEK(36867)
460 P=(P)AND254
470 POKE 36867,P
480 P=PEEK(36866)
490 P=(P)OR128
500 POKE 36866,P
510 POKE 36869,253
520 RETURN
530 PRINT"□";
540 DIM A(7,7)
550 GOSUB1130
560 REM PRINT MATRIX
565 REM ---------------
570 PRINT"■";
580 FORI=0TO7
590 FORJ=0TO7
600 A$=CHR$(A(J,I))
```



The VIC Magician

by
Michael S. Tomczyk

"LEARNING ABOUT THE CURSOR"

The subject of this "magical" installment is how to position or "program" information to print where you want it on the screen. This covers everything from how the cursor works to how to write programs that print words or graphics in specific locations on the screen.

Cursoring Around

Cursor control is one of the VIC 20's most powerful features. (The cursor is that blinking square on the screen that tells you where the next symbol will appear).

There are many ways to move the cursor around the screen, to make it appear and disappear and do crazy things . . . but the cursor's real power is its ability to *position* graphics, letters, numbers on the screen. Let's explore the cursor in depth and see how it works.

When you first turn on the computer, you should see a blinking blue rectangle (the "cursor") directly below the opening display. The cursor is controlled by the CRSR keys.

CRSR Keys

The CRSR keys are located at the lower righthand corner of your keyboard. They're the ones with the arrows on them. As you can see, these two keys let you move the cursor to the right or left, up or down. If you press the [ ] key, the cursor moves *down* the screen. If you hold down the SHIFT key and press the *same key* the cursor moves *up* the screen.

Moving the cursor right and left is just as easy. The [ ] key moves the cursor to the right and SHIFTing the same key moves the cursor to the left.

Special Features of the VIC 20 Cursor

Here are some special features of the VIC 20 cursor controls:

1. Automatic repeat. If you hold down either of the two CRSR keys you'll discover that the cursor automatically *repeats* as long as you hold it down. This is to help you move quickly to a desired location and is a powerful "screen editing" feature of the VIC 20.

2. Scrolling. If you press the "down" cursor key and keep holding it, you'll see that the cursor moves to the bottom of the screen . . . and when the cursor hits the bottom the entire screen will *scroll up one line at a time*. This is to give you more space when you're writing a program. Note that the VIC 20 screen only scrolls when you "cursor down." Moving the cursor to the top of the screen does not have any scrolling effect. Normally, scrolling only occurs when you move the cursor *down*.

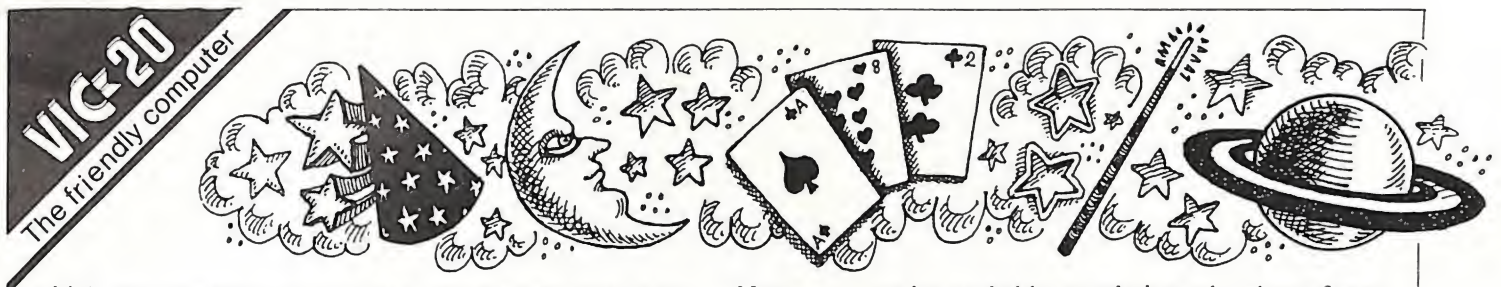
3. Wraparound. Try moving the cursor *horizontally* by pressing the CRSR RIGHT key. Notice that when it reaches the end of the line you're on, it automatically *jumps down to the beginning of the next line*. Conversely, if you SHIFT CURSOR LEFT the cursor will move left and jump *up* one line when it reaches the edge of the screen. This process of jumping up or down from one line to the next is called "wraparound."

4. CLR/HOME. Often, you want to move the cursor to the top lefthand corner of the screen. This is called the "home" position. The HOME key on the VIC is located at the top right corner of the keyboard. If you hit this key the cursor moves "home." If you hold down the SHIFT key and type CLR/HOME, you are actually typing *CLEAR* which *erases* any information you might have on the screen and positions the cursor at the home position.

To see how these keys work, try typing some information on the screen. Now type the HOME key. The cursor jumps to the "home" position. Now type CLEAR by holding down the SHIFT key and typing CLR/HOME. All information is gone and the cursor is in the 'home" position.

5. RETURN/SHIFT. You can also move the cursor down the lefthand column by hitting the RETURN key . . . but be careful when using this method. As a computer, the VIC has been taught to read and understand computer *programs*.

→



which are identified by typing a line number from 0 to 65000 in the far lefthand column of the screen. If you type a word without any line number and that word is not one of the "commands" in the VIC's vocabulary, the VIC will tell you that you've made a *programming error*. Here's how it works . . .

Try this: Hit the CLR/HOME key and type the word HELLO. Now hit the RETURN key. The VIC responds by telling you you've made a SYNTAX ERROR. This makes it difficult to type several lines on the screen. One way to overcome this is to type your message and then hold down the SHIFT key and hit the RETURN key. The cursor will move to the next line but the HELLO command will not be "entered" and the VIC will not give you an error message. To try it, type HELLO, then hold down the SHIFT key and hit RETURN.

This SHIFT/RETURN key combination can be a very powerful feature. If, for example, you're drawing a graphic picture . . . you can draw the picture and move to the next line quickly using SHIFT/RETURN without getting any error messages. This is helpful because a common technique in creating graphic programs in BASIC is to first draw the graphic picture on the screen, then add line numbers, quotation marks and the PRINT command along the lefthand column to convert your picture into a numbered BASIC program. (See the COLOR & GRAPHICS chapter in the VIC owner's manual). SHIFT/RETURN is also useful for moving around a BASIC program displayed on your screen when you want to move to different areas for editing purposes without affecting the program lines.

Programming the Cursor

So far, we've discussed some ways to move the cursor in *direct mode*. Now let's see how you can move and position the cursor in your computer *programs*.

You can PRINT cursor commands inside your computer programs—just like letters, numbers and graphic symbols. The format for doing this is exactly the same as PRINTing any VIC character. Try typing this program line:

```
10 PRINT "HELLO"
```

Now press the [CRSR RIGHT] key five times between the first quotation mark and the word HELLO. If you type this it will appear on your screen like this:

```
10 PRINT "     HELLO"
```

(Don't forget to hit RETURN at the end of the line to enter it).

Don't worry about the reverse bracket signs. We'll explain those in a moment. Now type RUN and hit RETURN. In the previous example, the word HELLO was printed in the left column. Now the word is printed 5 spaces over to the right because we put five CURSOR RIGHT commands in our program. The VIC moved the cursor five spaces from the left column and printed the word HELLO, just as it was instructed.

Now . . . you're probably wondering why those funny brackets appeared when you typed the CURSOR RIGHT key. The VIC uses special reverse symbols to show you where cursor commands are located in your program. This is helpful when editing a program or studying a program you haven't seen before. Here's a list of the graphic symbols used to represent the various cursor keys:

CURSOR RIGHT
CURSOR LEFT
CURSOR UP
CURSOR DOWN
HOME
CLEAR

The key to positioning a word or graphic image somewhere on the screen . . . or even making it MOVE in animated programs . . . is using the CURSOR key with the PRINT statement. Here are some exercises to give you some practice:

Exercise 1.
Type the same HELLO program except use CURSOR UP instead of CURSOR DOWN. This is an interesting one.

Exercise 2. **This is one of the most common programming techniques.**

```
10 PRINT "[CLR/] HELLO"
20 FOR X=1to1000
30 PRINT "[CLR/] BYE"
```

Exercise 3. **Try combining the CLEAR and CURSOR RIGHT commands.**

```
10 PRINT" ]]]HELLO"
20 FOR X = 1 TO 1000
30 PRINT" ]]]]]]]GOODBYE"
40 FOR X = 1 TO 1000
50 GOTO 10
```

Exercise 4. **Here's another version of Exercise 3 using just the CLEAR command.**

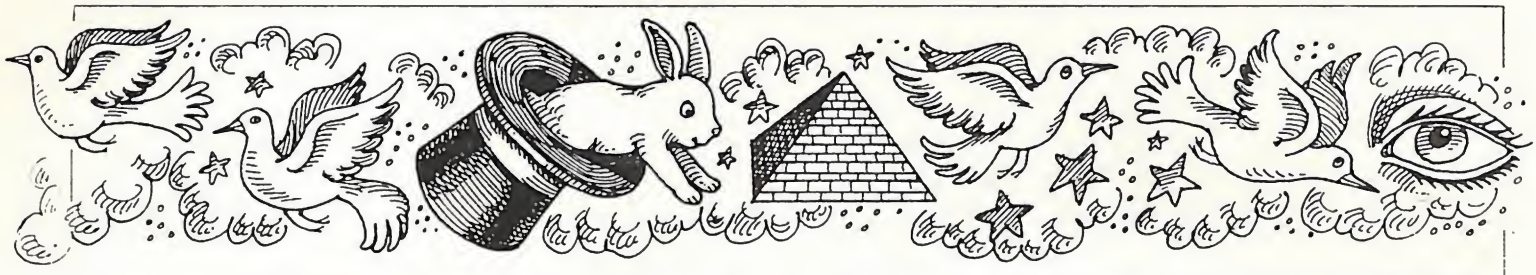
```
10 PRINT"[CLRHME]HELLO"
20 FORX=1TO1000:NEXT
30 PRINT"[CLRHME]GOODBYE"
40 FORX=1TO1000:NEXT
50 GOTO10
```

Exercise 5. **A simple animation example.**

```
10 PRINT"CLRHME O":FORX=1TO150:NEXT
20 PRINT"CLRHME ]0":FORX=1TO150:NEXT
30 PRINT"CLRHME ]]00":FORX=1TO 150:NEXT
40 GOTO10
```

Moving the Cursor With CHR$ Codes

Having learned that the cursor can be included in PRINT statements like any VIC character, it stands to reason that cursor commands would have their own CHR$ codes so you can use them in CHR$ statements, like VIC characters



The format CHR$(90) provides a more powerful alternative to the PRINT statement when displaying and manipulating VIC characters. All characters, including function keys and cursor controls, have their own CHR$ code numbers. For example, the number for the letter "Z" is 90, so if you type the following command, the letter Z will be printed on the screen:

```
10 PRINT CHR$(90)
```

This looks more clumsy than simply PRINTing the letter Z, but in many instances you can't or don't want to use the PRINT statement, so you use CHR$. In any event, here are the CHR$ numbers for the cursor controls:

CHR$ CODE	SCREEN MOVEMENT	
29	CURSOR RIGHT	The format for using
157	CURSOR LEFT	this technique in a
145	CURSOR UP	program is to type:10
17	CURSOR DOWN	PRINT CHR$(29)
19	HOME	
147	CLEAR	

Exercise 6.
Here's an example of how you can print a "CLEAR" command using the CHR$ technique:

```
10 PRINT CHR$(147)"VIC CLEARS, MOVES HOME
   AND PRINTS MESSAGE"
20 PRINT CHR$(17)CHR$(17)CHR$(17)"NOW
   DOWN"CHR$(17)
30 PRINT CHR$(29)CHR$(29)CHR$(29)"THEN
   RIGHT"CHR$(17)"OKAY!"
```

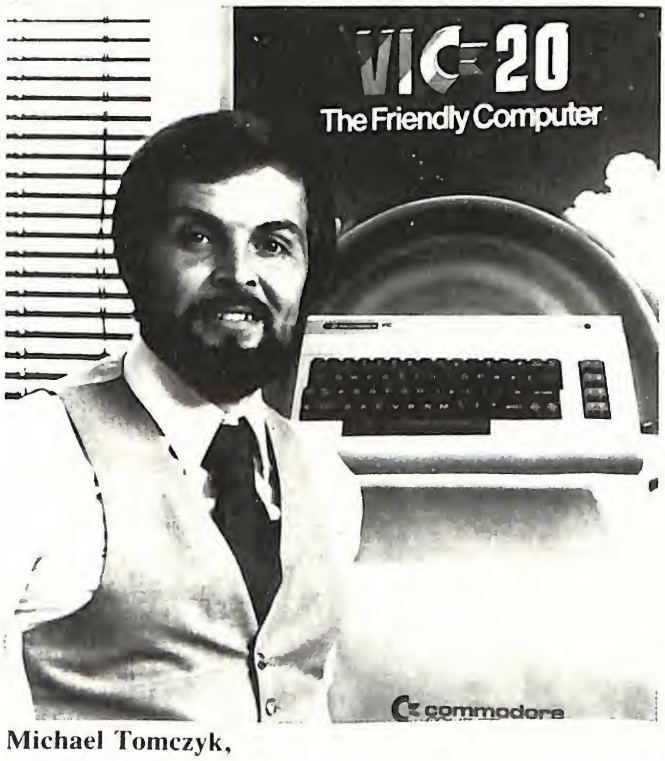
Things to note in this example include how the CHR$ statements are placed after the PRINT command. . . how in line 20 information printed in quotes can be mixed with CHR$ statements . . . how several CHR$ statements can be printed in a row to move the cursor more than once . . . how different messages and CHR$ statements can be "mixed and matched" as in line 30.

Exercise 7.
You can use "variables" in CHR$ statements, for example if you're going to be using the statements several times in a large program. Variables are important and we'll do a future article on them, but for now think of a variable as one or two letters which can be used as a *substitute* for a number, word, sentence or other piece of information. In this case, we will begin by "defining" our variable A equal to 147 (CHR$ code for CLEAR). This means we can substitute the letter A for the number 147 in my program. Note that we can still use the letter A normally in words and sentences, and that we can still use the number 147 if we want. This example simply clears the screen and prints HELLO.

```
10 A=147
20 PRINTCHR$(A)"HELLO"
```

These few examples were designed to help you understand how the VIC 20's *screen editing* commands work, especially in your programs. It's one thing for a computer to be as flexible as the VIC in placing information on the screen, but it's equally impressive that you can *write these positioning commands in your BASIC programs!*

The VIC "MAGICIAN" is a continuing series of how-to articles designed to take new VIC owners several steps beyond the VIC 20 Owner's guide. If you would like to learn more about programming the VIC 20, Commodore provides several excellent learning tools . . . including the VIC 20 PROGRAMMER'S REFERENCE GUIDE and the TEACH YOURSELF PROGRAMMING SERIES of books and tapes. Commodore also provides several special "computing aid cartridges" including the VIC 20 SUPEREXPANDER CARTRIDGE, VIC PROGRAMMER'S AID CARTRIDGE, and VICMON machine language monitor. ■



Michael Tomczyk,

COMMODORE COMES OF AGE.

APL now available

by Dick Van Digglen.

Commodore, always in the forefront of the microcomputer revolution, have now released their latest and best machine, the SuperPET SP9000. This new model has been built around 2 microprocessors, the 6502 which has been standard in Commodore for several years, and the 6809, a faster and more powerful chip which makes this machine more versatile than older Pet's and it's competitors. The SuperPET is switchable from 6502 to 6809 by the use of a simple toggle switch on the side of the cabinet.

In 6502 mode, the standard PET BASIC with about 32K of user area is available. In 6809 mode a menu designed by Waterloo Computing Systems is shown. The user is then allowed to select APL, BASIC, FORTRAN or Pascal. The requested programming language is then loaded from the system diskette in Drive 1, Drive 0 being available for a data diskette. After about one minute the machine is available for use and, in the case of APL, the user is presented with a clear workspace and about 29K of work area.

APL HISTORY

APL, as a computer language, is unique and thought by many to be the best currently available. The main reason for this is that it was originally designed as a consistent, unifying system for mathematical notation rather than as a computer language. It was invented by Kenneth Iverson in the late 1950's and derives it's name from a 1962 book called "A Programming Language". However, it wasn't until 1966 that an experimental time-sharing system for the IBM System/360 became available. Later implementations of the language were made by Scientific Time Sharing Corporation with APL*PLUS, IBM with APLSV and VSAPL and I.P. Sharp with it's version of APL.

Implementations for microcomputers have only just been made available, probably due to the fact that APL with a small work area is effectively useless. Certainly there would be little point having APL available on an 8K micro. The first major "micro" APL was IBM's 5100. Recently, the Intertec Superbrain was made available with a version of APL written by Telecompute Integrated Systems and now Commodore SuperPET joins the ranks with it's version written by Waterloo Computing Systems.

WHY USE APL ?

APL is a language which is so versatile it can be used as effectively by scientists as by businessmen. It has a great range of internal functions as well as good file handling. MicroAPL, as implemented on the SuperPET, follows closely the IBM standard for all it's workspace operations but makes use of a file handling system similar to that used in the I.P. Sharp network. IBM processes external files by use of shared variables, a facility which is not as easy to use as the Sharp system. the SuperPET allows access to the floppy disk drives for I/O and comes equipped with some utility APL workspaces for such operations as disk verification and disk to disk copy. In addition, the printer is viewed by APL as another output file which can therefore be easily opened for output when hard copy is desired.

So, what is so special about APL ? APL is the only language which frees the user from worrying about the internals of the machine and gives him the minimum number of restrictions. If you assign the value 3 to a variable called A we call this implicit definition of the variable, a facility which was freely available in FORTRAN. However, unlike FORTRAN, the value of A can be changed at any time from numeric to character and from a single number to a 3 dimensional array !

Also, consider two 2 dimensional arrays called A and B. If you wished to add the corresponding elements of each array in a BASIC program, you would write a double nested loop with, say, the inner loop indexing across the columns and the outer loop indexing down the rows. In APL, you just say A+B and let the machine worry about the looping. Similarly, A+10 will add the constant 10 to every element of the variable A.

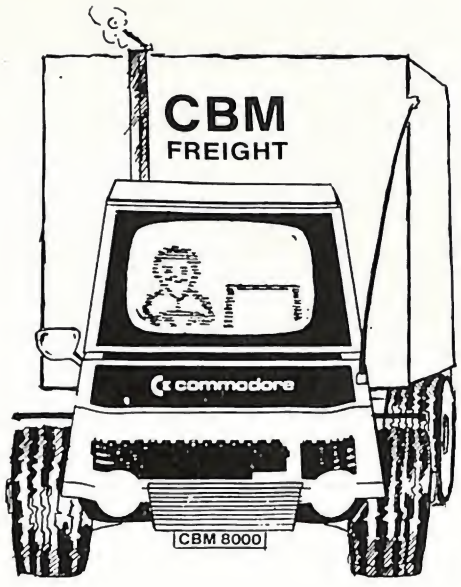
There are restrictions to such operations, however. If A has 3 rows and 5 columns and B has 7 rows and 4 columns, the result of A+B is LENGTH ERROR as APL has no idea how we wish these two matrices to be added. In all cases, APL will perform an operation which appears logical and could be viewed as the only sensible thing to do. There are no exceptions.

APL also has more operators than any other language. This is not to say that one needs to know all of them to get started with the language, but the great number of operators makes programming very simple. Examples of APL operators range from simple arithmetic, logarithms, exponentiation and the trigonometric functions to matrix inversion, complex character manipulation and finding the solutions to similtaneous linear equations. In addition, the method by which APL is programmed encourages the use of functional subroutines which the programmer can use to build up a library of "building blocks" from which he can easily assemble new programs.

APL is fast developing as the language of the future. It is consistent from one machine to another which will enable anyone who learns on the SuperPET to have no difficulty in programming in APL on any other machine at a later date. We can expect to hear more about it very soon.

Dick Van Digglen will be conducting APL Training Courses in the near future. For further details write to:-

Ragua Pty Ltd,
27 Rutherford Avenue,
Cronulla.
N.S.W. 2230.



EBS TRANSPORT SYSTEM PACKAGE

INTRODUCTION

The EBS Transport System was designed to fill the accounting needs of small transport and parcel delivery companies.

From a single key stroke entry of the delivery docket details for each carrying job, all files are similtaneously updated.

The system maintains records for Customers, Depots, Other Carriers and either Employee Drivers or Sub-Contractors.

SYSTEM BENEFITS

•* Reduces by approximately ⅔ the labour needed to manage the company's accounting – no need to sort delivery dockets by customer – resulting in cost reduction.

• Customer Statements can be posted out in a timely manner thereby improving cash flow.

CUSTOMER STATEMENTS

At any time Customer Invoice/Statements can be produced automatically detailing every transaction charged to that customer for the current month. The transaction detail shown on the statement provides up to 4 descriptive lines for each transaction , identification of the Driver, Depot, Other Carrier (if the goods were shipped part way by some other carrier and their freight charge is to be included in the account), sender, receiver, number of parcels And weight, A sample Invoice/Statement is attached. Pre-printed forms are also available.

The information given on the Invoice/Statement is adequate proof of the work carried out and can be further validated if necessary by reference to te original delivery docket showing the receiver's signature. It is therefore no longer necessary for the transport company to attach to the customer statement copies of each individual delivery docket as is the standard accounting practice in the industry.

The time saved which would otherwise be spent sorting delivery dockets by customer is considerable and typically amounts to around 40 man hours per month for a small operation.

SETTLEMENT REPORTS

The system also provides automatically calculated Settlement Reports for;

1. Sub-Contractors
2. Depot Report
3. Other Carrier Report

TRANSACTION LISTINGS

Trasaction can be listed for the current month sorted into customer order and grouped by Customer, Sub-Contractor, Depot, Other Carrier or simply listed in transaction reference number order.

CUSTOMER LISTS

Customer lists are provided in both summary form, where the details for each customer take up one line only, as well as full detail where all information kept by the system is presented for each customer. These reports are available in customer number or alphabetic sequence.

AGING REPORT

The Customer Aging Report details Customer Number, Customer Name, Telephone Number, Account Balance, Current Balance, 30 Day Balance, 60 Day Balance and Grand Totals for the customer accounts included in the report. Customers to be included in the report can be selected by a nominated minimum balance outstanding and by age of this balance. For example only customers who owe $50.00 or more that is 60 days or older might be included in a particular report. The report can be printed in customer number or alphabetic sequence.

OTHER REPORTS

An obligatory audit trail is printed whenever data held on file is to be added to or changed. This report picks up the old debtors Ledger Total, adds or subtracts the data entry or change and produces then shows the new Debtors Ledger Total. A Banking Report can be optionally printed following the Cash Receipts Report.

EBS DEBTORS DATA CAPTURE PACKAGE

INTRODUCTION

The EBS Debtors Data Capture System comprises a Commodore VIC 20 Computer, Commodore Cassette Tape Recorder, any T.V. set either black & white or colour and the EBS Data Capture Software Package.

Data is entered by the user, at their office and stored on tape. The tapes are periodically (e.g. weekly) processed by an arranged Bureau Service using the powerful Commodore 8000 series equipment which stores the user's complete Debtor's records on magnetic disk.

Using the VIC equipment the customer can construct customer records with name, address, phone number & other information. They can enter Salesman and Product descriptions and all transaction information comprising invoices, credit notes, cash receipts and miscellaneous adjustments. Sales tax, distributions of values to Salesman and Product Records can also be entered.

To streamline data entry, the system can be told not to ask for certain information such as sales tax, sales distributions and early payment discount offers to the user's customers if these facilities are not required in his particular type of business.

All the reports and statements provided by the EBS V38050 software on the Commodore 8000 equipment are available to the EBS Debtors Data Capture System user. These include Customer Statements, Aging Reports, Sales Tax Reports, Customer Lists and Audit Trail of data entered, in addition to full error reporting of any information have created an invoice for a customer whose record has not been set up.

Accurate Bureau Service billing is assured because the system automatically generates a service invoice based on the terms and rates agreed between the use and the Bureau Operator.

Other EBS Data Entry Software Packages will be available soon.
HARDWARE

HARDWARE

At the User End – VIC 20 Computer in it's basic form; Commodore Cassette Tape Drive; Any balck & white or Colout T.V. Set; One EBS Program Tape; A supply of blank tapes for data capture.

At the Bureau End – Commodore 8032 Computer; Commodore 8050 Disk Drive; Any printer interfaced to the 8032; Commodore Cassette Tape Drive. EBS V38050 Debtors System & Designer Disk Software.

EBS PUBLIC ACCOUNTANTS PACKAGE

INTRODUCTION

This is a low cost, efficient, computer based system, designed to fill the needs of the Public Accountant whose practice is small to medium in size.

The function of the system is to produce all necessary Financial Reports following the entry of each transition for income and expenditure in the current accounting period. Monthly reporting can be provided if desired and transactions can be listed monthly or yearly and sorted into groups under each account heading. The system provides an Assets Register with automatic calculation and reporting of Depreciation Schedules and Investment Allowance entitlements available at any time. An unrestricted number of Livestock Trading Accounts can be set up to fill the requirements of Farmer & Grazier Clients.

In the case of a client business which has several facits of opertaion or departments, the system can process the accounts of each part of the businss individually, providing full financial reporting and then consolidate all departments of the business operation into one set of financial statements for the overall business operation. Two levels of Consolidation can be utilised.

Either one of the two powerful Word Processing Systems available on Commodore will supplement this General Ledger System by automating the production of Directors' Reports and any other "letter writing" applications in the Accountant's office.

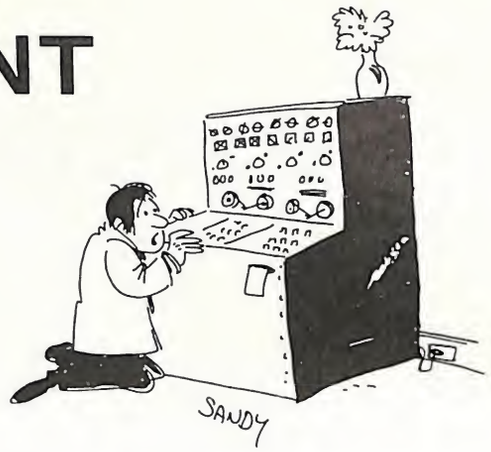
FEATURES

The main features of the system are:-

- Low cost
- Labour cost reduction in preparation of clients' accounts - resulting in increased productivity, lower cost per job and increased profit for the Accounting Practice.
- Elapsed job time reduced – resulting in Taxation Department deadlines being more easily met, timely completion of client financial reporting, earlier client billing improving cash flow...
- Typographical errors eliminated resulting in further reduction of elapsed job time.

LIFE WASN'T MEANT TO BE EASY

BY NICKI SAUNDERS



"Forgive me, Betty, forgive me! I'm sorry I kicked you in the side panel..."

Have you ever wondered what life was like hundreds of years ago ? In the days when man led a simple existence with no modern conveniences. I guess the nearest thing which came close to a computer was the Abacus. In fact, we probably have the Chinese to thank for our computer technology of today. Throughout history though, man has attempted to devise various means to improve his lot. Leonardo di Vinci not only painted the famous 'Mona Lisa' but also delved into aeronautical engineering. Men such as he, with clever minds and highly tuned imaginations, have given us what we have today – the Computer Revolution. However, not all modern technology has been good to us. Who can say that we have benefitted by the discovery of the Atomic Bomb ? The people of Hiroshima certainly didn't !! There is an old saying ' a little knowledge is a dangerous thing '. Ignorance can be blissful.

Being human though, I am also a hypocrite. Whilst I do not criticise the introduction of the car, the washing machine or the telephone, I can attack the computer. I find them somewhat the curse of my life. I think the difference is that the car, washing machine and telephone are tools and only used as such. Whilst although the computer is also a tool, it seems that anyone who operates it has a paranoic desire to enter and become one with the machine. I wonder what Freud would have said had he been alive today ?

This brings me to my husband, Bill. I have a very strong suspicion that he is having wild and passionate affairs with our computers. He never seems to tire of them. All day is not enough – he has to return at night 'til the early hours of the morning. He tinkers with RAMS and ROMS, he PEEKS and POKES, he INTERFACES – he is never sated.

Some of you may know that Bill and I have a Commodore Dealership. It all began over two years ago – a day which became a great turning point in our lives. Women all over the world have managed to cope with the 'other woman', the 'ternal triangle' but the 'other computer'....I ask you !!!! Yes, this is my lot but I am fascinated by my competitor. The caniving tricks it performs are veritably hitting below the belt. Have you ever witnessed a VISICALC demonstration, the success of 'on-line' to America, or the magic fingertip commands of WORDCRAFT ? These are what I have to compete with – an impossible task and I've tried. Boy, have I tried !!!

Over the years, I have discovered I am not the only one. Not only fellow dealers have had their marital problems because of this capricious involvement but also customer's wives. Some women become involved with their dearly beloved's PET. Others are totally repulsed. But, you must admit, it can be frustrating to find floppy disks littering you linen cupboard. 'What are these ?', you casually ask, 'Masters !' comes the curt reply. A fait-accompli. I know of a woman who has lost her spare bedroom to a computer system. Whilst she worries what will happen if anyone wants to stay, he justifies it, relishing the fact that they can't house the 'mother-in-law'.

Our customers come from all walks of life. Whether the computer be used as a hobby or for dedicated business applications, a games disk is vital at installation. 'For the kids you know' is the classic excuse. Kids, poor devils, they're lucky to get a look in.

There is no doubting that a computer makes a drastic change to anyone's life. Whilst he may stop the saturday night 'booze-up' with the boys, you still miss out on his company. He's invariably imprisoned in the den - 'Just making a few backups, dear'. This lasts for several hours. Perhaps he is doing calisthenics, you think as the occasional 'SHIT' echoes down the corridor.

I once rang Bill up at 2 a.m. to say he really ought to come home to bed. I got most stringently informed that I had interrupted a potent thought and would I please '...of'. I've stopped ringing.

Although I can be involved in various ways, I cannot, could and never will appreciate the subtleties of programming. An art truely in it's own field. Bill cries 'come and look at this', I dutifully appear to see a screen with a flashing cursor asking a question. Bill demonstrates a '5 second wonder'. I am honestly amazed. Whilst he treats my fascination and loud exclamations as further proofs of his manly superiorities, I think 'whoopee'. Poor Bill !!!!

Well I solved my loneliness problem somewhat by getting a Burmese kitten. Errol is a continual source of entertainment. But, I fear, he too has been bitten by the computer bug. He loves to watch the printer inaction and becomes glued to the flashing lights of the disk drive. He has even learnt that pressing the return key in the IMS Debtors Package produces an interesting bell-like noise. Errol has devised a few programs. They illogically consist of random input produced by walking over the keyboard. Although numerous 'systax errors' occur, I do believe Errol is working towards some unique conclusion.

Truely,

Life was't meant to be easy !!!!!

BUTTERFIELD ON COMMODORE

The Friendly PET — Screen editing

One of the friendliest things about the PET, CBM and VIC is the way they allow you to make a change or correction. If the line on the screen is wrong—whether it is a program line or a direct command—we can move the cursor back and type over the line. Pressing the RETURN key will make the change take effect.

Correcting Programs.
This is very handy for programs. When your first program attempts result in a message such as ?SYNTAX ERROR IN 350 you can list 350 to see the problem. If line 350 happens to say PWINT X, you can move the cursor back, type R over the W to give PRINT X, and strike RETURN. The line has been corrected with a minimum of typing on your part.

If you need to make an insertion into your program, you may use the INSERT key. If the mistake was PINT X, the technique is to position the cursor over the I, hold down the SHIFT key, and press INST for insert; the computer will open up space and you can type in the missing R. On the other hand, if the error was PHRINT X you'll want to make a deletion: place the cursor over the R, press the DEL key to delete, and the H will disappear. In either case, don't forget to press RETURN to make the change permanent.

If you hapen to goof in making the change, start over. In this case don't press RETURN. Hold down SHIFT and then press RETURN: this will take you to the next line without any program change being made.

Shifted-RETURN is quite a handy key combination to know for many reasons. If you wanted to leave a note on the computer's screen for someone to read, you might type MARY - PUT THE CAT OUT. At this point, striking RETURN would cause the computer to try to "perform" the line, and you'd get ?SYNTAX ERROR. If you press

Shifted-RETURN, however, you'll just go to the next line and the computer won't try to do anything with the contents of the previous line.

The INSERT key has some special rules. After you have pressed the INSERT key a number of times (don't forget to hold down SHIFT) there will be an open space on the screen where you can insert the new characters. At this point, you'll be in "programmed cursor" mode. This means that the cursor keys don't move the cursor; instead they will print as special reversed characters. This is the same way that the PET behaves after you press the quote-mark key, with two important exceptions: the computer remains in this mode only for the number of characters to be inserted; and the Delete (DEL) and Insert (INST) keys work in a different way. More about this another time; in the meantime, you'll get used to them quite quickly.

A problem sometimes crops up if a program line is too long. Sometimes this means that there's no extra space available to make a desired insertion—eighty characters is the screen limit. Worse, the line is too long to start with; it occupies over 80 characters even before we make a change. It might be more sensible to change it to two lines and relieve the crowding; but if you must, the trick is to look through the line to find a keyword that can be abbreviated. PRINT is the most popular, since it can be rewritten as a question mark. Close up the space, making sure that everything is packed into the 80-column work area, and then make the change if it fits.

The Direct Approach.
Direct lines—Basic commands typed in without a line number so they are executed right away—are usually easy to fix. If you mistype LOAD "PROGRAM" so that it comes out LOUD "PROGRAM", don't be dismayed by the ?SYNTAX ERROR. You can slip the cursor back, change the U to an A, press RETURN and the load will take place.

Correcting mistakes in Direct lines can leave a cluttered screen. When I try to load BOTTLESHIPS from the disk, I get several lines which tell me that there's no such program. When I move the cursor back and correct to BATTLESHIPS, the following lines don't go away unless written over. It looks messy, but works OK.

There's a sharper problem when I ask a direct statement to print a number. If I ask the PET to calculate 4*5*6, yielding a product of 120, and then decide that I really want addition, I can go back and change the asterisks to plus signs. The PET will now produce a total of 15, but the last digit of the previous answer won't be wiped out; the Zero will be left on the screen and our sum will look like 150 instead of 15. The solution? Wipe out any numbers you want recalculated so that the new values will print on a fresh line.

Special screenings.
When you press RETURN, the PET sees only what's on the screen. You may have done deletions, insertions, and changes but the final screen result is all that counts. This is true of program lines, direct commands, and responses to program INPUT statements.

You may want to run a program several times while testing, with similar answers to INPUT questions on each run. With screen editing, it's a snap. After you have run once, move the cursor back to the RUN statement. Press RETURN (no need to type RUN: it's on the screen). For each INPUT, the cursor will appear over the answer you typed on the previous run. If you want to go with the same response this time, just press RETURN and the program will accept the same input from the screen. If you want to change, type your new input.

Here's a hint of advance techniques that you'll learn as you become more familiar with your computer. You can actually get the PET to type its own input—even its own program changes—to the screen. Then, with a stroke of the RETURN key, you can activate the input or program change. When used for INPUT activities, this provides a "default" input for the user. As a program changes, the program could suggest DATA statements that it would like to see included in a future run. Mind boggling! At this rate, the computer could program itself and make us all obsolete.

At least, the computer still needs us to press the RETURN key; it can't do that by itself. Or can it? Technical tyros suggest that POKE 158,1:POKE 623,13 (or on Original ROMs, POKE 525,1:POKE 527,13) would actually cause the PET to send a carriage return to itself. . .

First Programming Steps

The first programs that a beginner writes tend to be simple. That's good, of course: the programmer is developing skills which will be useful when he tackles more ambitious jobs. Here are a few suggestions on how to go about these early projects; the emphasis will be more on sound practices and clear style rather than clever coding methods. Some of the suggestions might be useful for experienced programmers, too.

Try to lay out your programs in "blocks". Each block should have a clear, simple function. One block might do an input job, another might calculate, and a third generate output. If you start planning a program by thinking out the blocks you will need, your program will be better planned. Some programmers make each block into a subroutine so that the main program simply calls in these units as needed.

Title each section or block with a remarks REM statement. You don't have to put comments on each line, but it's useful to be able to find a section of code quickly. Perhaps you think that you can remember the code—after all, you wrote it—but wait a couple of months. It's amazing how a crystal clear progam can suddenly become gibberish after you've been away from it for a while. Leave yourself some highway markers so that you can find your way around later.

Name your variables in a semi-meaningful way. Totals can start with the letter T, counts with a C, and so on. I'm not a fan of large alphabetic names, since they have pitfalls:

TERRIFIC is a great label, but it doesn't work since the keyword IF is hidden in the middle. Can you find the hidden keywords in GRANDPA, CATNIP, CRUNCH and FRONT? It's fun to play word games, but not when you're trying to write a program. I prefer a single letter followed by a numeric: T4, B7 and so on. By the way, don't forget that variable B has nothing to do with integer variable B% or string B$ or for that matter array variable B(3). They are all completely independent values.

Don't let anyone hustle you about program size or speed. If others write in less memory and fewer milliseconds, let them. You'll have space enough for most of your programs and the tenth of a second saved in run time won't give you time for a cup of coffee. On the other hand, do look for better methods. Better isn't always faster or smaller, but you'll recognize it when you see it.

Keep track of your variables; it's useful to make a list on a sheet of paper. That way, you won't accidentally use variable X for two different jobs and get them mixed up. In fact, it doesn't hurt to do paperwork planning before turning your computer on. There's a kind of "heat" in working directly on the machine that sometimes leads to hasty programming. A little leisurely planning beforehand can generate sounder and better programs.

Don't be afraid to write loosely. The fanatic who tells you that you'll save memory and time by compacting FOR M = S TO P STEP V into FORM=STOPSTEPV is steering you wrong in most cases. If legibility costs you four bytes and one millisecond, take it: it's a bargain.

If your program doesn't work right the first time, don't lose heart. It happens to most of us. The easy errors are where the computer tells you where the problem is, most commonly ?SYNTAX ERROR IN . . . The problem will likely be obvious when you look at the line; if not, you can try rewriting it slightly to see what happens. The hard errors are where the computer doesn't stop, but gives you the wrong answers.

Debugging can be great fun if you can take the right attitude. Look at the variables: you can call them up with direct PRINT statements. Change them if it suits your purpose. Put STOP commands into your program and check everything out when you come to the halt. You can resume where you left off with CONT. Using the RUN/STOP key to break your program in mid-execution is less precise but will also do the job.

Getting a program together can be a rewarding experience—not necessarily rewarding in money, but in a sense of accomplishment. Each program will be a work of art, done in your own style. When you put your signature to your latest masterpiece, you'll feel good about it if you've used good coding craftmanship.

Half a dialogue — Inputting

Asking a porgram to go and get input from the user is a subtle thing for beginners. When you write your first programs, it's hard to look ahead and see the program independently communicating with the user. "If the program needs a value, I'll program it in right now . . ." It takes a level of sophistication to imagine a program accepting working values at a later time, when it runs, and using different values supplied by the user in different runs.

There are three fundamental ways of checking what the user is doing at the keyboard: INPUT, GET, and a PEEK. We'll talk about each, and its uses.

INPUT.
The INPUT statement does a lot of work for you. It's certainly one of the most powerful statements in BASIC. Some of us would like to see it more powerful, and some would like to see it less sophisticated; for the moment, we'll have to accept it as it is.

When you give the command INPUT in a program, a prompting question mark is printed and the cursor begins to flash. Your program is held in suspended animation; it will not resume operation until RETURN is pressed. There's no code which allows something like:

INPUT M:IF (NO REPLY IN 15 SECONDS) GOTO . . .

Your code will hang on the INPUT statement forever if the user doesn't reply.

When the user presses RETURN, INPUT takes the information from the screen. It doesn't matter if the user wandered back and forth, changing, deleting and inserting; INPUT looks only at the screen which is the result of his/her actions. In fact, if there's something on the screen that the user didn't type, INPUT will take that too. This can be useful for prompting: you can arrange to type a sample response on the screen, and the user will be able to press RETURN to have that response entered. As INPUT takes the information from the screen, it trims away all leading and trailing spaces; other than that it takes the whole line, even though it may not need it.

Now INPUT starts to plow through the line, digging out the information you need for your program. If it's looking for a number it will not like to find a string, and will ask, REDO FROM START. If it's looking for a string, it won't mind a number at all: it will accept it as a string.

Road signs for INPUT.
Whether INPUT is looking for a number or a string, it will stop its search when it finds one of three things; comma, colon, or end of line. If it finds a comma it will assume that more information will be needed later in the INPUT statement; if it finds a colon or end of line it assumes that there is no more useful input from the user. If it needs more, it will ask for it.

Suppose you need to input a string that contains a comma or a colon, such as ULYSSES M PHIPPS, PHD. or ATTENTION: JOHN, MARY. Since INPUT normally stops at the comma or colon character, we need to do something. The answer is easy: the user must put the desired input in quotes: "ATTENTION: JOHN, MARY" and the whole thing commas, colon and all, will be received as a single string.

Keep in mind that the INPUT statement allows prompting. INPUT "YOUR NAME";N$ causes the computer to type YOUR NAME? and wait for input. That's a good human interface: helping the user along.

If a user presses RETURN without supplying any information on the screen, programs on the PET/CBM will stop. There are several ways to prevent this from happening; the easiest is to add a "canned reply" to the input prompt message. When you are writing the INPUT statement prompt (such as YOUR NAME) add two extra spaces and, say, an asterisk character; then type three Cursor-Lefts (they will print as an odd-looking reversed bar) and close the quotes on the prompt. Finish the INPUT statement in the usual way: a semicolon behind the prompt and then the name of the variable to be input. Now: the asterisk or whatever will print to the right of the prompt and question mark. Unless the user overtypes it, this character will be received from the screen as input—and the program won't stop.

One last comment: don't forget that INPUT can accept several values. You can say INPUT N$,A$,C$ and allow the user to type JOE BLOW, CITY HALL, DENVER. It's often better to use separate input statements: users can respond better when prompted for each piece of information.

GET and PEEK: a preview

GET isn't as clever as INPUT, but it has valuable uses. First of all, it doesn't wait; if a key isn't ready in the keyboard buffer, the GET statement lets BASIC continue. Secondly, keystrokes received with GET don't affect the screen unless you, the programmer, decide to allow them to do so. This means that you have much more control over what the user can do.

There's a PEEK location (PEEK(151) on most PET/CBMs, PEEK(515) on Original ROMs, and PEEK(197) on the VIC that tells you whether a key is being held down or not. This can be useful to avoid the situation where a user needs to press the same key repeatedly to cause some action; you can program so that the key repeats its action if it is held down.

We'll talk in more detail about the GET and PEEK next time around. They are more fun in some ways than the INPUT statement . . . but they call for quite a bit more programming work to be done.■

MACHINE LANGUAGE
Scanning the Stack

Jim Butterfield.

The stack is a group of locations from hexadecimal $0100 to $01FF that can be quickly and conveniently used by the 6502. In the PET, the stack range is limited to the area from $0140 to$ 01FA; but most of the time you don't need to know where the stack is working; you may just use it.

When you have something that needs keeping for a few moments, you can put it on the stack and call it back later. So long as you're neat, you don't even need to know where it will go – the processor keeps track of that with a special register called a Stack Pointer. It will put information away to the stack and bring it back without any special information from you.

But you must be neat. The slogan "Leave these premises as clean as you found them" applies critically to the way you use the stack. If you put something in there, you must be sure to call it back or you'll be in trouble.

The stack is appropriately named. It's like a stack of dishes: the first thing that comes off will be the last thing that was put on. It's called LIFO (Last-in-first-out) storage.

Standard usage

If you a value in the A register that you want to put aside for a moment or so, you can push it to the stack using the PHA (48) instruction. Now you can use the A register for something else, and when you're finished you can call back the original value by pulling it from the stack with PLA (68).

Sometimes you might want to defer a decision. You've just done a comparison or some other activity, and the results are important - but you don't want to act on those results yet. You can push the status word – all various flags, such as Carry, Overflow, etc. – to stack with PHP (08). Now you can tidy up your registers without worrying about those losing flags. They will come back as soon as you give PLP (28) and you can then proceed with the Branch commands that will test the condition you previously set up.

When you call a subroutine with a JSR command, the stack is called into play automatically. The return address, minus one, is placed on the stack. Later, when the RTS is given, that address is called back from the stack and program execution resumes at the instruction following the JSR.

An example here might be worth while. If you are at location hex $1234, and give the instruction JSR $4455, the address $1236 will be placed on the stack. That's not your return point – you'll return to $1237 since the JSR command is 3 bytes long – but the RTS instruction will

sort everything out correctly. Here's a little more detail: when the address $1236 is placed on the stack it will use two locations. The high-order part (12) goes onto the stack first, followed by the low-order portion (36).

When the 6502 receives an interrupt – and on this PET this happens 60 times a second – the current machine language instruction completes; the address of the next instruction is pushed to the stack; and finally, the processor Status Word is pushed to the stack. Then the processor starts to handle the interrupt by going to a new location and executing instructions there. When it's finished, it gives a Return from Interrupt instruction (RTI) which restores the original Status Word and instruction address. The original program picks up exactly where it left off when it was interrupted.

You can see that three locations are used in the stack this time: two for the return address and one for the status word. They go onto the stack in that order: address-high, address-low, and status. It's a little like JSR followed by a PHP, since we store address and status word. Note, however, that the address is the exact return address; with a JSR the address is one less than the return address.

An example: if the processor is executing a three-byte instruction at hex $1234 and an interrupt is signalled, address $1237 is pushed to the stack, followed by the status word. Later, when RTI is executed, the status word is restored and execution resumes at address $1237.

Finally, the BRK instruction (hex $00) causes an interrupt type of action, with this difference: the address which is placed on the stack is two locations behind the Break instruction. This is odd, since the BRK command is only one byte long. In this case, if we use and RTI to continue executing the code following the BRK, we'll skip one byte.

Tabular Summary

The following table summarizes the instructions that handle the stack.

Number of bytes stored or recalled	*Store command*	*Recall command*
1	*PHA*	*PLA*
1	*PHP*	*PLP*
2	*JSR*	*RTS*
3	*interrupt*	*RTI*
3	*BRK*	

Here's where it gets interesting. "Ordinary" programming assumes that you use the companion instruction to restore the stack. That is, if you used a PHA you should use a PLA to bring the information back. If you used a JSR, you should use an RTS. But by breaking this unwritten law, we can do some pretty fancy things. We must be careful, of course.

The Fun Begins

Suppose you are writing a subroutine. Normally, you'll want to return control to the calling point by giving the RTS command. On occasion, however, you don't want to go back; perhaps there's an error in the data so that the calling routine couldn't continue.

We can handle this. Just pop the return address from the stack with two PLA commands , and you'll never go back.

A little more detail on the tricks we can use here. Suppose you have a main routine at A, which calls a subroutine at B. Subroutine B, in turn, calls subroutine C several times. Subroutine C, which might for example be digging out a parameter for the SAVE command, decides it doesn't want to go back to subroutine B for some reason; perhaps there are no more parameters left (e.g., SAVE "PGM" instead of SAVE "PGM",1,2). In this case, it wants to go straight back to the main routine A.

When subroutine C is called, the stack will contain four values: two for the return address to A, and two for the return address to B. If subroutine C executes: PLA PLA RTS, it will throw away the return to B and go straight back to A.

Decimal Quickie

Want to find out if you're in Decimal mode ? It's unusual in the PET, but the 6502 processor can switch to a special mode for addition and subtraction. There's a flag in the status word that signals this. You could try a sample addition and see whether the result is calculated in decimal or not: CLC – LDA #$05 – ADC #$05 .. the result will be hexadecimal $0A if you're in binary mode, and hexadecimal $10 if you are in decimal mode.

There's a more straight-forward way. Push the Status Word to the stack, and pull it back to the A register – execute PHP, PLA. You cn now examine the bits of the Status Word at your leisure. Decimal mode is flagged in bit 3; you could mask it with AND#$08, for example.

The Computed Jump

You can jump to any single location you choose by using the JMP instruction. There are times, however, when you want to jump to one of several locations depending on some value you have calculated.

For example, you might be writing a system which would jump to one routine if it detected an A (add) character; another routine for D (delete); a third for C (change); and so on.

This could be done, of course, with a series of compare, branch and jump instructions; but if the list is long, the whole thing becomes tedious and inefficient.

You can set up the equivalent of a very powerful computed jump by clever use of the stack. The principle is manufacture an address; push it to the stack with PHA ... PHA; and then give RTS.

This seems puzzling at first. However can you return to a place you never came from ? It works this way: by pushing the address to the stack, you stimulate a non-existent subroutine call. The stack doesn't care. If you issue an RTS instruction, the stack will deliver up that address, and that's where you will go. The stack ends up unchanged: it has pushed two values and delivered them back.

Remember that the RTS instruction expects the address to be one lower than the real return address. If you want to go address hex $3456, you must push the values 34 and 55.

A quick example may help illistrate this powerful technique. Suppose the X register contains a value from 0 to 5. Depending on this

value, we wish to jump to one of six different locations. We have built the destination addresses into a set of address tables, each with six entries. The low order part of the addresses are in a table starting at hex $2320 and the high order part of the addresses are in a second table starting at hex $2326. We've carefully remembered to subtract one from each address, and the table looks like the following:-

```
2320 41 72 A3 C4 E5 F6
2326 24 25 27 29 2B 2C
```

If X contains zero, we want to jump to hex $2442; if one, we go to $2573; and so on. Let's do it.

```
BD 26 23  LDA $2326,X ; high order first
48        PHA
BD 20 23  LDA $2320,X ; low order last
48        PHA
60        RTS          ; go there
```

It's easy, it's fast, it's compact, and it's one of the most powerful tricks in the repertoire of the 6502 programmer. Microsoft Basic uses it to get to the various – PRINT, LET, FOR ... etc. The Machine Language Monitor uses it to interpret it's commands – M.R., and so on.

The Stack Pointer

Scanning the Stack

The Stack Pointer

There are a couple of commands that tell you where the stack is working, to allow you to control where it is. You won't need to use them very often, but a little detail is worthwhile.

The stack works in a downwards direction. As you push things, the stack pointer gets lower. As you pull them back, the pointer goes back up. If the stack pointer gets down to its bottom value, pointing at address hex $0100, it will wrap around to $01FF if you try pushing more things in; but your program will be in serious trouble long before you reach this point.

The stack pointer indicates the next location that will be used. If the pointer has a value of 92, you know that the next value that you push will go into address $0192; or the next value that you will pull will come from address $0193.

The command TXS – Transfer X to Stack Pointer, hex $9A – is most often used to reset the stack completely. This cancels everything: subroutines, interrupts, the whole works. On most PET's, you should set the pointer to hex $FA with: LDX#$FA, TXS.

The command TXS – Transfer Stack Pointer to X, hex $BA – is used for a couple of things.

You can check to see if you have too many things on the stack by coding TSX, CPX#$40 ... use whatever pointer limit you think is reasonable, but hex $40 is about as low as you should ever allow it to get.

The stack is in memory, of course; so you can look through the stack directly by examining the contents of locations $0100 to $01FF. You'll need to know where to start, of course, and TSX comes in handy here. If you give TSX followed by LDA $0100,X you will load the location to which the stack points. That may not be too useful, since it's the next location to be filled, and isn't part of the "active" stack. By incrementing X, however, or by looking higher with an instruction such as LDA $0101,X you can get to whatever part of the stack interests you.

Summary

Most of the time, the stack will take care of itself. Occasionally, however, you'll find that digging a little deeper into the mechanics of the stack can make it possible to do some very effective coding.

---

continued from page 5

APPLE
1802 (with Super BASIC)
PET/CBM
SWTPC-6800
TRS-80
OSI-1P
P2000 (Phillips)
ATOM ACORN
DAI
EXIDY SORCERER
VIDEO GENIE

... and there are more computers to join !

The translation program is specific to a specific brand of computer. For some brands a small piece of electronic circuitry is also necessary.

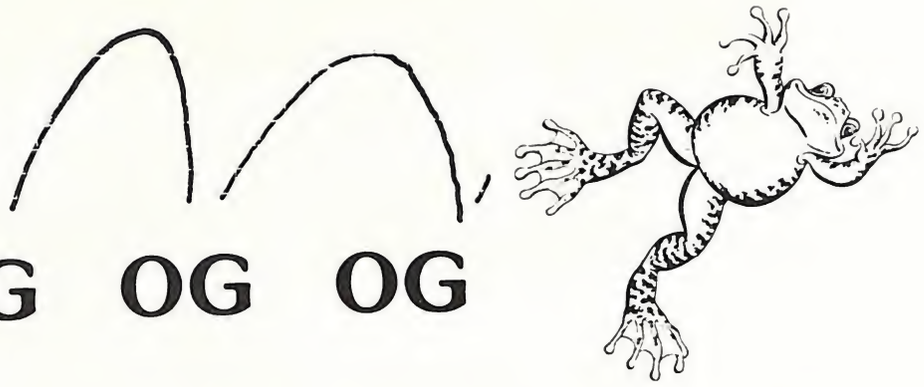
The initiative to BASICODE was taken by Hans G. Janssen and Klaas Robers. They founded the NOS-Hobbyscoop BASICODE Group. This group defined the BASICODE Communication Standard and developed the translation programs for the computer brands mentioned above. The Dutch Educational Computer Groups TEACHIP and DIDACOM are also members of the BASICODE Group; they are developing the BASICODE programming standard.

All information on BASICODE, translation programs included, will be published in a special BASICODE MANUAL by the NOS. The computer specific translation program and examples of programs in BASICODE are assembled on a cassette tape. This tape will be issued together with the BASICODE manual. This will be in April 1982. The book will be written in English and Dutch.

Dr R. Docters Van Leeuwen

(NOS stands for Netherlands Omroep Stichting: Dutch Broadcasting Foundation).

(NOS – Hobbyscoop is a Dutch radio program which transmits computer programmes in BASICODE by AM and FM, thus establishing a completely new computer data network in Holland.)



LEAPFROG OG OG

by P.R. Gabor.

A few words 80 column users who want to see what's happening. Before typing the program in, and indeed before running it thereafter, type:-

printchr$(142)

followed by pressing the

'RETURN' key.

This puts your PET into graphics mode, and also gives you continuous graphics on the screen, rather than a slight gap between each line. To get out of this afterwards, use 14 instead of 142. A number of graphics characters are used in the listing, and to save you spending hours looking for them (since they're not actually marked on the keyboard), here are the relevant characters:-

In line 110 we encounter most of the graphics characters producing cursor movement. So line 110 would, in descriptive form, read:-

110 F$(0)=" (3c1)(cd) (cu)"

The only other one we come across occurs first of all in line 150, where the second character inside quotes is a cursor right. Now for the graphics characters. To define these, if a character in a line has a '-' over it, press SHIFT as well as the relevant character to get the desired graphical result. There are seven characters used in all, and line 130 features 5 of them. In descriptive form this would read:-

130 F$(2)="J(rvs)Q(off)K(3cl,cd)U(rvs)Q(off)I(cu)

These are not exactly like the characters in the listing, but they're the closest the 8032 can get, since the characters listed, used to be produced by shifting the numeric keys, and of course this doesn't work anymore. The other characters we encounter are in line 150 first of all, where the first character within quotes is a shifted C, and in line 260 where the line is achieved by pressing SHIFTed R (again this is as close as 8032 users can get, as the line should really be a shifted $ sign, but all we get is ... a $ sign !)

THE GAME

Pleasure can be mixed with business and the following program allows Commodore users to fiddle with the various features of the PET and learn a little bit of BASIC programming.

This is a simple checker-game, called LEAPFROG. The object of the game is to exchange the places of the frogs, as explained in lines 300 to 440.

Line 460 waits for an input to start the game. The first problem was to make a graphical image of the frogs, I think the shifted 'Q' characters are very suitable for this.

The actual game board is the array A(X), a value of '1' represents a green frog, a '2' a black frog and a zero an empty space.

Now the main problem was to write an intelligent input sequence, where nothing but the appropriate number keys should be pressed. This meant using 'get' and not 'input' and that the PET had to be able to differentiate whether a '1' meant 'one' or whether it was the first digit of a two-digit number. A '0' could not be accepted as the first digit and the 'delete' key should annul the entry.

The input routine starts at line 1340, the actual 'get' routine at line 1820. Note the use of the flag DEL. The reason for it is that we cannot jump directly from the subroutine to the 'delete' routine. First we have to return to the calling routine (terminating the subroutine sequence)- and the calling routine must be told it had to start the input routine all over again.

The second digit input might require some additional explanation. Line 1850 checks if the first digit is a '1'. If not, then there is no second digit. Line 1860 multiplies contents of elements 8-11 of array A(X). Now one of these elements must be a zero (pointing to an empty space) if the leap is to or from a two digit number. Therefore if the result of this multiplication is not zero, there is no second digit.

By the way, line 1900 could reject a second digit greater than one (change '9' to '1')- but I could not resist the temptation to have a frog 'scream' and fall off the screen. (The animated sequence is between lines 1600 and 1760). Incidentally, the regular leaps are animated in lines 710-810, lines 730, 750, 770, 800 are the time delays between the phases of the leap, these of course can be adjusted as required.

A time delay in lines 1650 and 1660 (A=sin(3)) can also be adjusted.

I believe the rest is properly explained by the remarks in the listing.

So type in the program and have fun. Incidentally – it can be done in 35 moves.

```
10 REM ********************************
20 REM *                              *
30 REM *      L E A P F P O G         *
40 REM *                              *
50 REM *         BY P. GABOR          *
60 REM *           20/08/81           *
70 REM *                              *
80 REM ********************************



90
100 DIM A%(19)
110 F$(0)="   ■■■■    "                    REM     ARRAY FOR FROGS
120 F$(1)="  ◣■■■■◢ ◣■■■■◢"                REM     SPACE
130 F$(2)=" ◣■■■■■■■■■■■◢"                REM     STRING TO PRINT GREEN FROG
140                                        REM     BLACK FROG
150 AN$(1)="■■■■■■■■" AN$(2)="■■■■■■■■"
160 AN$(3)="■■■■■■■■" AN$(4)="■■■■■■■■"    REM     STRETCH LEGS
170 AN$(5)=F$(0)                           REM     PREPARE FOR TAKEOFF
180 AN$(6)="■■■■■■■■"                      REM     AND OFF YOU GO
190                                        REM     FALLING!
200 PP$="■■■■■"                            REM     MOVE CURSOR TO FIFTH ROW
210
220 REM ********************************
230 REM *         THE RULES            *
240 REM ********************************
250
260 T1$="                    "
270 T1$=T1$+CHR$(13)+"                "
280 T1$=T1$+"■■ LEAP - FROG ■■"
290 PRINTT1$ PRINT
300 PRINT"FIVE LITTLE GREEN FROGS AND FIVE LITTLE"
310 PRINT"BLACK FROGS ARE SITTING ON A VERY TALL"
320 PRINT"FENCE. THERE IS JUST ROOM FOR ONE FROG"
330 PRINT"BETWEEN THEM. LIKE THIS  -■"
340 FOR I=1TO5 PRINTF$(1), NEXT PRINT"    ".
350 FOR I=1TO5 PRINTF$(2), NEXT PRINT"■■"
360 PRINT"THE OBJECT OF THIS GAME IS TO EXCHANGE"
370 PRINT"THE GREEN AND THE BLACK FROGS BY MOVING"
380 PRINT"ONE FROG AT A TIME BY ONE SPACE (IF THE"
390 PRINT"NEXT SPACE IS FREE) OR HAVING A FROG"
400 PRINT"LEAP OVER ITS NEIGHBOUR TO A FREE SPACE"
410 PRINT"(DON'T LEAP OVER MORE THAN ONE FROG)."
420 PRINT"TO REACH THIS POSITION  -■"
430 FOR I=1TO5 PRINTF$(2), NEXT PRINT"    ".
440 FOR I=1TO5 PRINTF$(1), NEXT PRINT"■■"
450 PRINT"■■    ■PRESS ANY KEY TO START THE GAME"
460 GET A$  IF A$="" THEN 460
470
480 REM ********************************
490 REM *        START GAME            *
500 REM ********************************
510
520 FOR I=1TO5                             REM     INITIALIZE ARRAY
530   A(I)=1 A(I+6)=2
540 NEXT A(6)=0
550 C=0                                    REM     SET COUNTER TO ZERO
560 PRINTT1$ PRINT PRINT
570 FOR I=1TO11                            REM     PRINT FROGS
580   PRINTF$(A(I)),,
590 NEXT PRINT"■■"
600 PRINT"_____________________________"
610 PRINT"■■ 1  2  3  4  5  6  7  8  9  10 11"
620 PRINTPP$"■■■■■"
630 PRINT"ENTER YOUR MOVE!    - "
640 GOSUB1340 FF=0                         REM     RESET FLAG FOR FALL-OFF
650 PRINT"■" TI=1500                       REM     SET CONSTANT FOR TIME DELAY
660 IF A(S)=0 THEN 1470                    REM     STARTING POS. EMPTY
670 IF ABS(S-E)<2 THEN 1500                REM     LEAP TOO LARGE
680 IF A(E) THEN 1520                      REM     ENDPOSITION OCCUPIED
690 IF E<11 THEN FF=1  E=12                REM     JUMP OFF BOARD
700  =0
710 PRINTPP$,TAB(3*(S-1)),,                REM     POSITION TO JUMPING FROG
720 FOR I=1 TO 5                           REM     ANIMATE JUMP
730   FOR L=1 TO 25  NEXT
740   PRINTAN$(I),,
750   FOR L=1 TO 60 NEXT
760 NEXT
770 FOR L=1 TO 50 NEXT
780 PRINTPP$,TAB(3*(E-1)),,F$(A(S)),,
790 PRINT"■■■"AN$(I),,
800 FOR I=1 TO 150 NEXT
810 PRINTF$(A(S)),,
820 C=C+1
830 A(E)=A(S)  A(S)=0
840 FOR I=1TO6
850   I=I+A(I)*10^I
860 NEXT
870 IF FF THEN 1600
880 IF INT(C/2)=C/2 THEN 620               REM COMPUTED C IS NOT EXACT INTEGER
890
900 REM ********************************
910 REM *          FINISH              *
920 REM ********************************
930
940 PRINT"■■■■■■" IF C<60 THEN990
950 PRINT"_____________________________"
960 PRINT"■YOU FINISHED, AT LAST!!!"
970 GOTO1020
980
990 IF C<50 THEN1060
1000 PRINT"_____________________"
1010 PRINT"■NOT A BAD RESULT!!!"
1020 PRINT"■YOU NEEDED"C"MOVES TO SOLVE THE PROBLEM"
1030 PRINT"YOU SHOULD REALLY DO BETTER!!"
1040 GOTO1240
1050
1060 IF C<40 THEN1120
1070 PRINT"_____________"
1080 PRINT"■VERY WELL DONE!!!"
1090 PRINT"■YOU SUCCEEDED TO COMPLETE THE GAME IN"
1100 PRINT"ONLY"C"MOVES - AN ABOVE AVERAGE RESULT!" GOTO1240
1110
1120 IF C=35 THEN1190
1130 PRINT"_____________________________________"
1140 PRINT"■■EXCELLENT!! YOU ARE A REAL EXPERT!!!"
1150 PRINT"■YOU HAVE DONE IT IN ONLY"C"MOVES. THIS"
1160 PRINT"IS ALMOST THE BEST POSSIBLE RESULT."
1170 GOTO1240
1180
1190 PRINT"_____________________________________"
1200 PRINT"■■CONGRATULATIONS! NOBODY CAN DO IT BETTER"
1210 PRINT"■YOU COMPLETED THE GAME IN 35 STEPS."
1220 PRINT"THIS IS THE ABSOLUTE MINIMUM."
1230
1240 PRINT"■■WOULD YOU LIKE TO PLAY AGAIN? (Y/N)"
1250 GET A$  IF A$<>"Y" AND A$<>"N" THEN1250
1260 IF A$="Y" THEN520
1270 PRINT"■THANKS FOR PLAYING  LEAP-FROG  -"
1280 END
1290
1300 REM ********************************
1310 REM *       INPUT ROUTINE          *
1320 REM ********************************
1330
1340 POKE158,0                             REM     NO CHAR IN KEYBOARD BUFFER
1350 DEL=0 PRINT"■"                        REM     FLAG, SET IF DELETE
1360 PRINTTAB(23)"FROM          ■"
1370 PRINTTAB(28),"■ ■■".
1380 GOSUB1820  IF DEL THEN 1350
1390 S=VAL(C$)  PRINT" TO ■ ■■".
1400 GOSUB1820  IF DEL THEN 1350
1410 E=VAL(C$)  RETURN
1420
1430 REM ********************************
1440 REM * MESSAGES FOR BAD INPUTS  *
1450 REM ********************************
1460
1470 PRINT"THERE IS NOBODY AT POSITION #"S"■■■"
1480 PRINT"PLEASE TRY AGAIN!" GOTO1570
1490
1500 PRINT"HEY, I CANNOT JUMP THAT FAR!!" GOTO1570
1510
1520 PRINT"HEY, WHAT DO YOU THINK YOU ARE DOING?"
1530 PRINT"■THAT IS MY BEST FRIEND, SITTING ON"
1540 PRINT"■SEAT NR."E"■!! YOU CAN'T EXPECT ME TO"
1550 PRINT"■LAND ON HIM!?!  PLEASE TRY AGAIN"
1560 TI=3000                               REM     INCREASE TIME DELAY
1570 FOR I=1 TO TI  NEXT                   REM     TIME DELAY
1580 GOTO560
1590
1600 PRINTPP$,TAB(33),F$(0),"■■■■".
1610 PRINTF$(A(E)),,"■■■■".AN$(1),,AN$(2),,
1620 A$="■■■■■■■■"
1630 PRINT"■■■■",TAB(34),,
1640 FOR I=1 TO 20
1650   PRINT"■"A$,  A=SIN(3)
1660   PRINT"■"A$,  A=SIN(3)
1670 NEXT  PRINT"       ■■■■■"
1680 PRINT"_____________________________"
1690 PRINT"■NOW LOOK WHAT YOU HAVE DONE!!"
1700 PRINTPP$,"■",TAB(34),,AN$(0),,
1710 FOR I=72 TO 0 STEP -4
1720   PRINTF$(A(E)),,"■■■■".AN$(0),,
1730   FOR L=1 TO I  NEXT
1740   PRINTF$(0),,"■■■■".
1750 NEXT A(12)=0
1760 PRINTF$(0),,"!!!!!!" GOTO 1240
1770
1780 REM ********************************
1790 REM ♦GET A ONE- OR TWO-DIGIT NR ♦
1800 REM ********************************
1810
1820 GETC$  IF C$="" THEN 1820
1830 IF C$=CHR$(20) THEN DEL=1 RETURN      REM     DELETE
1840 IF C$<"1" OR C$>"9" THEN 1820         REM     REJECT ENTRY
1850 PRINTC$;  IF C$<>"1" THEN RETURN      REM     SINGLE DIGIT ENTRY
1860 IFA(8)♦A(9)♦A(10)♦A(11)THENRETURN     REM     POS 8-11 OCCUPIED
1870 PRINT"■ ■■".
1880 GETCC$ IF CC$="" THEN 1880
1890 IF CC$=CHR$(20) THEN DEL=1 RETURN     REM     DELETE
1900 IF CC$<"0" OR CC$>"9" THEN 1880       REM     REJECT ENTRY
1910 C=C*10+CC$ PRINTCC$, RETURN
READY.
```

RediForm

instant business forms and ancillary products for the small computer user.

Rediform is a totally new concept in the supply of forms requirements for the small computer user. No longer do new users need to wait for delivery of important computer forms.

Your Commodore Dealer who has supplied the software package can also supply the relevant pre-set forms, off the shelf, along with other start-up requirements.

Rediform offers a range of...

Pre-set instant business forms:

Rediform pre-set carbonless forms designs are specially designed for common business applications such as invoicing, accounts payable, accounts receivable etc., allowing every business to take advantage of expert and accurate forms designs approved by computer suppliers.

Stock computer print-out paper:

Rediform carbonless print-out paper is available in convenient carry packs in a range of sizes- either singles or multi-part. The carry pack doubles as a storage module for permanent filing of print-out.

In addition, Rediform offers a range of Self-Adhesive Continuous Labels, Forms Files, and Forms Handling Equipment.

Rediform product and information is available from your Commodore Dealer or contact

Moore Paragon Australia,
offices in all capital cities.

RediForm

SPEL RITE

by Bob Friend and Tony Ellis.

This program is intended as a spelling primer.

The words to be learned are entered as data statements from line 500.

Incorrectly spelled words are also entered in the data statements for comparison.

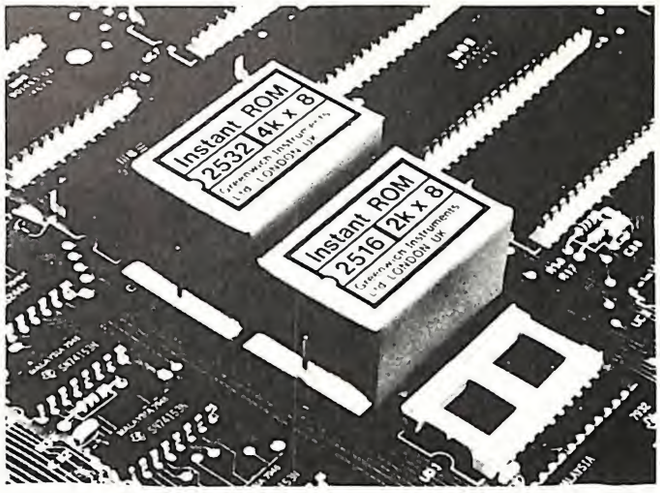
The words are selected randomly and the student is asked if the spelling is correct and to spell the word if they think it is incorrect.

A running tally of the number of correct answers is kept, however this is reset if the program is listed which is a partial check against cheating.

The data is restored after each word so the same words may appear more often than others but the program keeps running until stopped.

If a longer list of words is used, lines 110 and 190 would need adjustment.

```
10 PRINT"□":PRINT:PRINT:PRINT:PRINTTAB(11)"''SPEL RITE''":PRINT:PRINT:PRINT
15 PRINT"LERNIN TER SPEL CORECTLEE"
16 FORI=1TO5000:NEXT
20 PRINT"□":PRINT:PRINT:PRINT"YOU WILL BE SHOWN A WORD,DECIDE IF IT IS"
30 PRINT"CORRECT, THEN ANSWER 'Y'OR'N'. YOU WILL"
40 PRINT:PRINT"THEN BE TOLD IF YOU WERE RIGHT OR NOT."
50 FORI=1TO9000:NEXT:PRINT"□":PRINT:PRINT
60 PRINT"WHEN YOU ENTER A WORD IT MUST CONTAIN 12"
70 PRINT"LETTERS,SO ADD DOTS (..) UP TO THE STAR."
80 PRINT:PRINTTAB(24)"*"
90 PRINT"LIKE THIS   DOG........."
100 FORI=1TO10000:NEXT
110 PRINT:PRINT:PRINT:PRINT"TO ADD NEW WORDS TYPE 'LIST500-600'"
120 PRINT:PRINT"REPLACE THE WORDS YOU FIND THERE WITH"
130 PRINT:PRINT"YOUR NEW WORDS. EACH WORD MUST CONTAIN"
140 PRINT:PRINT"12 LETTERS SO ADD DOTS TO SHORTER WORDS"
141 PRINT:PRINT"THE WORD WHICH IS CORRECT GOES IN THE"
142 PRINT:PRINT"MIDDLE."
150 FORI=1TO15000:NEXT
160 S=0:W=0
165 X=1:R=1:PRINT:PRINT:PRINT"□YOUR SCORE IS"S"OUT OF "W:FORI=1TO3000:NEXT
170 X=INT(RND(1)*3)+1
180 R=INT(RND(1)*10)+1
190 FORN=1TOR:READA$:NEXT
200 RESTORE
210 B1$=LEFT$(A$,12)
220 B2$=MID$(A$,13,12)
230 B3$=RIGHT$(A$,12)
240 PRINT"□":PRINT:PRINT:PRINT
250 IFX=1THENPRINTTAB(10)B1$
260 IFX=2THENPRINTB2$
270 IFX=3THENPRINTB3$
280 PRINT:PRINT:PRINT:INPUT"IS THIS THE CORRECT SPELLING";W$:W=W+1
290 IFW$="Y"ANDX=2THENPRINT:PRINT:PRINT"THATS RIGHT":S=S+1:GOTO380
295 IFW$="N"ANDX=2GOTO400
296 IFW$="Y"ANDX<>2THENPRINT:PRINT"NO ITS NOT TRY AND SPELL IT":GOTO310
300 PRINT:PRINT:PRINT"WELL SPELL IT FOR ME"
310 PRINT:PRINT:PRINTTAB(13)"*"
320 INPUTG$
330 IFG$=B2$THENPRINT:PRINT:PRINT"VERY GOOD":S=S+1:GOTO380
340 PRINT:PRINT:PRINT"SORRY, TRY AGAIN":W=W+1
345 PRINT:PRINT:PRINTTAB(13)"*"
346 INPUTG$
350 IFG$=B2$THEN PRINT:PRINT"GOOD RIGHT THIS TIME":S=S+1:GOTO380
360 PRINT:PRINT"NO,THE CORRECT SPELLING IS "B2$
370 FORI=1TO3000:NEXT
380 PRINT:PRINT:PRINT"HERE IS ANOTHER"
390 FORI=1TO5000:NEXT:GOTO165
400 PRINT:PRINT"WELL SPELL IT FOR ME":PRINT:PRINT:PRINTTAB(13)"*":PRINT:INPUTG$
410 IFG$=B2$THENPRINT:PRINT"YES THAT'S HOW I DID IT AT THE START"
420 PRINT:PRINT:PRINT"YOU SHOULD HAVE LEFT IT ALONE IT WAS OK":GOTO380
500 DATA"POBEET......POBEPT......PUBEPT......"
510 DATA"DAMPEM......DAMPEN......DAMEN......."
520 DATA"BLACKSLAND..BLAXLAND....BLUXLAND...."
530 DATA"COMISSION...COMMISSION..COMMISION.."
540 DATA"CAMBELLTOWN.CAMPBELLTOWNCAMPELLTOWN."
550 DATA"COLLIN......COLLEEN.....COLEEN....."
560 DATA"TECKNICIAN..TECHNICIAN..TECHNISHIAN."
570 DATA"PARRAMATTER.PARRAMATTA..PARAMATTA..."
580 DATA"MARSOUPIAL..MARSUPIAL...MASUPIAL...."
590 DATA"TRANSECTOR..TRANSACTOR..TRANSACTER.."
600 DATA"RECIEVE....RECEIVE.....RELIEEVE..."
```



HOW TO SWITCH OFF AND MAINTAIN MEMORY

AVAILABLE FOR: VIC/PET/CBM

What is it !

INSTANT ROM is 2K, 4K, or 8K of CMOS memory with battery back-up Plug it into the ROM expansion sockets on your VIC, PET or CBM, and write your program into it as if it were RAM. Your program is there, PERMANENTLY – for years! **Even when the power goes off!** You can erase a single byte, even a single bit – no erasure problems.

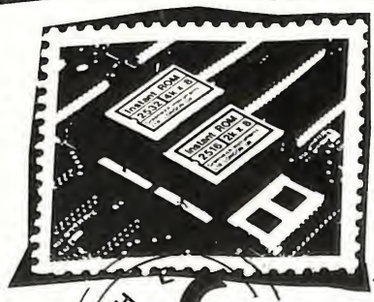
Put simply – Now you can write and modify your ROMs in minutes....

- ★ RE-WRITE BASIC
- ★ SWITCH-ON-AND-RUN BASIC PROGRAMS
- ★ YOUR OWN MACHINE LANGUAGE MONITOR
- ★ CUSTOM CHARACTER FONT
- ★ DEDICATED PROGRAMS
- ★ COMMUNICATION PACKS

MAIL ORDER

direct from the distributor;

CW ELECTRONICS
416 LOGAN RD.(Pacific Hwy.)
STONES CORNER, BRISBANE.
P.O. Box 274, SUNNYBANK Qld, 4109.
(07) 397 0808, 397 0888, telex AA40811



THE VIC CENTRE

INSTANT ROM			
2K byte ROM	GR2516	**$ 103**	
4K byte ROM	GR2532	**$ 139**	
8K byte ROM suitable for VIC 20	GR2364	**$ 219**	
	Freight	**$ 2**	**$ 2**
	TOTAL		

Please charge my BANKCARD No. ☐☐☐ ☐☐ ☐☐☐ ☐☐☐☐☐☐

Name..... Tel(w/h).....

Address .....

..... Postcode.....

COMMODORE BUSINESS MACHINES, INC.
PRODUCT SPECIFICATIONS

VIC 20

Screen size: 22 char. x 23 lines
Memory: 5K expandable to 32K
User supplied color monitor or TV
4 internal amplifiers including:
- 5 octaves total range
- 3-tone (music) generators
- 1 sound effects generator

Peripherals include: cassette recorder, printer, modem, disk drive, and IEEE-488 interface. RS-232 interface, memory expansion module, 3K-8K-16K expansion cartridges

Features:
- Programmable function keys
- Standard PET BASIC — upwardly compatible to other CBM computers
- Full-size typewriter keyboard
- Graphics character set, upper and lower case letters
- Plug-in program/memory cartridges
- Low-priced peripherals
- Joystick/paddles/lightpen
- Self-teaching materials
- 16 colors
- High resolution graphics (176 x 184 dots)
- Programmable characters
- Full screen editing

*Includes modulator

PET®

Screen size: 40 char. x 25 lines
Memory: 16K to 32K
IEEE-488 bus for disk, printer, modem and other intelligent peripherals
Eight-bit parallel user port with "handshake" lines
Supports Commodore C2N cassette and disk unit

Features:
- 128 ASCII plus 128 graphic characters
- 74-key professional keyboard
- Shift key gives 64 graphic characters
- Separate calculator/numeric pad
- Full screen editing capabilities
- 18K of ROM contains BASIC (version 4.0) with 9-digit floating binary arithmetic

CBM™ 8032 (Business Machine)

Screen size: 80 char. x 25 lines
Memory: 32K expandable to 96K
IEEE-488 bus for disk, printer, modem and other intelligent peripherals

Business software includes:
- Word processing
- VISICALC
- Inventory control
- Accounts receivable
- Accounts payable
- Payroll
- Dow Jones Portfolio Management
- Personnel files

Features:
- 128 ASCII, plus 128 graphics characters
- 73-key professional keyboard
- Separate calculator/numeric pad
- Full screen editing capabilities
- 18K of ROM contains BASIC (version 4.0) with direct (interactive) and program modes
- 9-digit floating binary arithmetic

SuperPET Computer

Screen size: 80 char. x 25 lines
Memory: 96K RAM (64K bank switched) IEEE-488 bus for disk, printer, modem and other intelligent peripherals High-speed RS232-C interface for additional compatibility with printer and modem
Communicates to Mainframes*
Compatible with 8032 software
2 microprocessors - 6502 and a 6809

Software includes: language interpreters, editor, operating system (supervisor), and an assembly language development system
- BASIC 4.0
- Waterloo MicroBASIC
- Waterloo MicroPascal
- Waterloo MicroFORTRAN
- Waterloo MicroAPL
- 6809 Assembler Development System, Linker/Loader

Features:
- 128 ASCII, plus 128 graphic characters, IBM/ACM, APL character set
- 73-key professional keyboard
- Separate calculator/number keyboard (line/text editor keypad usage in 6809 mode)
- Screen editing capabilities
- 38K of ROM contains BASIC (version 4.0) with direct (interactive) and program modes and 9-digit floating binary arithmetic

*Program file transfer between suitably equipped Mainframe. (Program upload/download between M/F and SuperPET). Programs will execute without modification.

commodore COMPUTER

3 Campbell Street, Artarmon,
N.S.W 2064, AUSTRALIA
Ph. (02) 437 6296